OpenSSH

en.wikibooks.org

June 25, 2024

On the 28th of April 2012 the contents of the English as well as German Wikibooks and Wikipedia projects were licensed under Creative Commons Attribution-ShareAlike 3.0 Unported license. A URI to this license is given in the list of figures on page 241. If this document is a derived work from the contents of one of these projects and the content was still licensed by the project under this license at the time of derivation this document has to be licensed under the same, a similar or a compatible license, as stated in section 4b of the license. The list of contributors is included in chapter Contributors on page 237. The licenses GPL, LGPL and GFDL are included in chapter Licenses on page 245, since this book and/or parts of it may or may not be licensed under one or more of these licenses, and thus require inclusion of these licenses. The licenses of the figures are given in the list of figures on page 241. This PDF was generated by the LaTeX typesetting software. The LaTeX source code is included as an attachment (`source.7z.txt`) in this PDF file. To extract the source from the PDF file, you can use the `pdfdetach` tool including in the `poppler` suite, or the `http://www.pdflabs.com/tools/pdftk-the-pdf-toolkit/` utility. Some PDF viewers may also let you save the attachment to a file. After extracting it from the PDF file you have to rename it to `source.7z`. To uncompress the resulting archive we recommend the use of `http://www.7-zip.org/`. The LaTeX source itself was generated by a program written by Dirk Hünniger, which is freely available under an open source license from `http://de.wikibooks.org/wiki/Benutzer:Dirk_Huenniger/wb2pdf`.

Contents

[1]

1 https://en.wikibooks.org/wiki/Help:Development_stages

1 OpenSSH/Print version

This is the print version[1] of OpenSSH[2]
You won't see this message or any elements not part of the book's content when you print or preview[3] this page.

OpenSSH

The current, editable version of this book is available in Wikibooks, the open-content textbooks collection, at

4

Permission is granted to copy, distribute, and/or modify this document under the terms of the Creative Commons Attribution-ShareAlike 3.0 License[5].

> *OpenSSH* is a featured book[6] on Wikibooks because it contains substantial content, it is well-formatted, and the Wikibooks community has decided to feature it on the main page[7] or in other places. Please continue to improve it and thanks for the great work so far! A `{{Goodbook`[8]`}}` template should be created to advertise it[9].

The OpenSSH suite provides secure remote access and file transfer.[1] Since its initial release, it has grown to become the most widely used implementation of the SSH protocol.

---

1 https://en.wikibooks.org/wiki/Help:Print_versions
2 https://en.wikibooks.org/wiki/OpenSSH
3 https://en.wikibooks.org/w/index.php?title=OpenSSH/Print_version&printable=yes
4 https://en.wikibooks.org/wiki/OpenSSH
5 https://en.wikibooks.org/wiki/Wikibooks:Creative_Commons_Attribution-ShareAlike_3.0_Unported_License
6 https://en.wikibooks.org/wiki/Wikibooks:Featured_books
7 https://en.wikibooks.org/wiki/Main_Page
8 https://en.wikibooks.org/wiki/Template:Goodbook
9 https://en.wikibooks.org/w/index.php?title=Template:Goodbook/OpenSSH/Print_version&action=edit&preload=Template:Goodbook/Blank&editintro=Template:Goodbook/Editintro

During the first ten years of its existence, it had largely replaced older corresponding unencrypted tools and protocols. The OpenSSH client is included by default in most operating system distributions, including MacOS, AIX, Linux, BSD, and Solaris. Any day you use the Internet, you are using and relying on hundreds if not thousands of machines operated and maintained using OpenSSH. A survey in 2008 showed that of the SSH servers found running, just over 80% were OpenSSH. [2] Even with the advent of the Internet of Things and the increased use of IPv6, a cursory search of Shodan [3] for SSH-2.0 services on port 22 in November 2022 showed 87% of responding addresses running OpenSSH. [4]

OpenSSH was first released towards the end of 1999. It is the latest step in a very long and useful history of networked computing, remote access, and telecommuting.

This book is for fellow users of OpenSSH to help save effort and time through using OpenSSH, and especially SFTP, where it makes sense to use it.

1. ”OpenSSH”[10]. www.openssh.com.
2. ”Statistics from the current scan results”[11]. OpenSSH.com. 2008.
3. ”SSH-2.0 Search Results”[12]. shodan.io. Retrieved 2022-11-07.
4. At that time, 87% was 17,093,847 out of 19,715,171 systems. Dropbear made up about 6% at 1,092,738 systems and then a long tail filled in the rest. A Shodan search including non-standard ports, not just port 22, showed noticeably more SSH services (25,520,277) whereof 83% (21,215,882) where OpenSSH.

Table of Contents

10 http://www.openssh.com/
11 http://www.openssh.com/usage/ssh-stats.html
12 https://www.shodan.io/search?query=ssh-2.0

2 Overview

The OpenSSH[1] suite provides secure remote access and file transfer. Since its initial release, it has grown to become the most widely used implementation of the SSH protocol. During the first ten years of its existence, ssh has largely replaced older corresponding unencrypted tools and protocols. The OpenSSH client is included by default in most operating system distributions, including MacOS, Linux, BSD, AIX and Solaris. Any day you use the Internet, you are using and relying on dozens if not hundreds of machines operated and maintained using OpenSSH. A survey in 2008 showed that of the SSH servers found running, just over 80% were OpenSSH. [1]

OpenSSH was first released towards the end of 1999. It is the latest step in a very long and useful history of networked commuting, remote access and telecommuting.

2.1 History of OpenSSH

The first release of OpenSSH was in December 1999 as part of OpenBSD 2.6. The source code was originally derived from a re-write of the last available open version, ssh 1.2.12 specifically, of SSH[2]. SSH went on to become Tectia SSH.

Ongoing development of OpenSSH is done by the OpenBSD group. Core development occurs first on OpenBSD, then portability teams bring the changes to other platforms. OpenSSH is an integral part of as good as all server systems today and a good many network appliances such as routers, switches and networked storage. The first steps were in many ways the biggest.

2.1.1 The Early Days of Remote Access

Some of the tools that inspired the need for SSH have been around since the beginning, too, or very near the beginning of the Internet. Remote access has been a fundamental part of the concept since the idea stage and the nature and capabilities of this access has evolved as the network has evolved in scale, scope and usage. See the web version of the *Lévénez Unix Timeline*[3] by Éric Lévénez for an overview of systems development and the web version of *Hobbes' Internet Timeline*[4] by Robert H Zakon for an overview of the development of the Internet.

1969

1 `http://www.openssh.com/`

- Telnet was one of the original ARPAnet application protocols, named in RFC 15[2] from September 1969. It was used to access a host at a remote site locally. Telnet was described starting two years later in RFC 137[3], RFC 139[4], RFC 318[5] and others, including RFC 97[6]. That is as good a turning point as any to delineate Telnet.

1971

- Thompson Shell, by Ken Thompson, was an improvement on the old text-based user interface, the shell. This new one allowed redirects but was only a user interface and not for scripting.
- In the same year FTP, the file transfer protocol, was described in RFC 114[7]. A key goal was to promote use of computers over the net by allowing users at any host on the network to use the file system of any cooperating host.

1978

- Bill Joy created BSD's C shell which is named for the C-like syntax it uses. It allows job control, history substitution, and aliases, which features we find in today's interfaces.
- In the same year, the Bourne Shell by Steve Bourne at Bell Labs [5] was created. It is the progenitor to the default shells used in most distros today: **ksh** and **bash**.

1983

- The remote file copy utility, **rcp**, appeared in 4.2 BSD. **rcp** copied files across the net to other hosts using **rsh**, which also appeared staring 4.2 BSD, to perform its operations. Like **telnet** and **ftp**, all passwords, user names, and data are transmitted unencrypted in clear text. Both **rsh** and **rcp** were part of the **rlogin** suite.

1991

- PGP, written at MIT by Philip Zimmermann[6], charted new waters for encrypted electronic communications with the goals of preserving civil liberties online, ensuring individual privacy, keeping encryption legal in the USA, and protecting business communications. Like SSH it uses asymmetric encryption with public / private key pairs.

1993

- Kerberos V (RFC 1510[8]) authentication service from MIT's project Athena [7] provides a means for authentication over an open, unsecure network. Kerberos got its original start in 1988.

2.1.2 SSH - open then closed

1995

2 https://tools.ietf.org/html/rfc15
3 https://tools.ietf.org/html/rfc137
4 https://tools.ietf.org/html/rfc139
5 https://tools.ietf.org/html/rfc318
6 https://tools.ietf.org/html/rfc97
7 https://tools.ietf.org/html/rfc114
8 https://tools.ietf.org/html/rfc1510

- Tatu Ylönen at the then Helsinki University of Technology developed the first SSH protocol and programs, releasing them under an open license[8] as per the norm in computer science, software engineering, and advanced development. [9]

1995?

- Björn Grönvall dug out the most recent open version of **ssh**, version 1.2.12[10] [11]. He and Holger Trapp did the initial work to free the distribution, resulting in OSSH

1996

- The SSH2 protocol is defined

2.1.3 OpenSSH

1999

- OpenSSH begins based on OSSH. Niels Provos, Theo de Raadt, Markus Friedl developed the cryptographic components during the port to OpenBSD which became the OpenSSH we know today. Dug Song, Aaron Campbell and many others provided various non-crypto contributions. openssl library issues were sorted by Bob Beck. Damien Miller, Philip Hands, and others started porting OpenSSH to Linux. Finally OpenSSH 1.2.2 was release shipped with OpenBSD 2.6 in December 1, 1999.[12]

2000

- Markus Friedl added SSH 2 protocol support to OpenSSH version 2.0, which was released in June.[13] OpenSSH 2.0 shipped with OpenBSD 2.7. Niels Provos and Theo de Raadt did most of the checking. Bob Beck updated OpenSSL. Markus also added support for the SFTP protocol later that same year.
- In September of 2000, the long wait in the USA for the patents on the RSA algorithms to expire was over. In the European Union the European Patent Convention of 1972 frees software, algorithms, business methods or literature, unlike the unfortunate, anti-business situation in the USA. This freedom in Europe hangs by a thread at the moment.
- SSH Tectia changes licenses again.

2001

- Damien Miller completed the SFTP client which was released in February.
- SSH2 became the default protocol

2008

- Built-in chroot support for **sshd**.

2010

- As of OpenSSH 5.4, the legacy protocol SSH1 is finally disabled by default.

2014

- As of OpenSSH 6.7, both the base and the portable versions of OpenSSH can build against LibreSSL instead of OpenSSL for certain cryptographic functions.

2016

- OpenSSH 7.4 removes server support for the SSH1 legacy protocol.

2023

- OpenSSH 9.5 ssh-keygen(1)[9] generates Ed25519 keys by default instead of old RSA keys.

Note: OpenSSH can be used anywhere in the whole world because it uses only algorithms unencumbered by software patents, business method patents, algorithm patents, and so on. These types of patents do not apply in Europe, only physical inventions can be patented in Europe, but there are regions of the world where these problems do occur. Small and medium businesses in Europe have been active in politics to keep the advantage.

2.2 Why Use OpenSSH?

A lot has changed since the commercialization of the Internet began in 1996. It was once a University and Government research network and if you were on the net back then, odds were you were supposed to be there. Though it was far from being utopia, any misbehavior could usually be quickly narrowed down to the individuals involved and dealt with easily, usually with no more than a phone call or a few e-mails. Few, if any, sessions back then were encrypted and both passwords and user names were passed in clear text.

By then, the WWW was more than a few years under way and undergoing explosive growth. The estimated number of web servers online in 1996 grew from 100,000 at the beginning of the year to close to 650,000 by the end of the same year[14]. When other types of servers are included in those figures, the estimated year-end number was over 16,000,000 hosts, representing approximately 828,000 domains.[14]

Nowadays, hosts are subject to hostile scans from the moment they are connected to the network. Any and all unencrypted traffic is scanned and parsed for user names, passwords, and other sensitive information. Currently, the biggest espionage threats come from private companies, but governments, individuals, and organized crime are not without a presence.

Each connection from one host to another goes through many networks and each packet may take the same or a different route there and back again. This example shows thirteen hops among three organizations from a student computer to a search engine:

```
% /usr/sbin/traceroute -n www.google.com
traceroute: Warning: www.google.com has multiple addresses; using 74.125.95.106
traceroute to www.l.google.com (74.125.95.106), 30 hops max, 40 byte packets
 1 xx.xx.xx.xx          0.419 ms    0.220 ms    0.213 ms    University of
 Michigan
 2 xx.xx.xx.xx          0.446 ms    0.349 ms    0.315 ms    Merit Network, Inc.
 3 xx.xx.xx.xx          0.572 ms    0.513 ms    0.525 ms    University of
 Michigan
 4 xx.xx.xx.xx          0.472 ms    0.425 ms    0.402 ms    University of
 Michigan
 5 xx.xx.xx.xx          0.647 ms    0.551 ms    0.561 ms    University of
 Michigan
 6 xx.xx.xx.xx          0.945 ms    0.912 ms    0.865 ms    University of
 Michigan
 7 xx.xx.xx.xx          6.478 ms    6.503 ms    6.489 ms    Merit Network, Inc.
 8 xx.xx.xx.xx            6.597 ms    6.590 ms    6.604 ms    Merit Network,
```

9 http://man.openbsd.org/ssh-keygen

```
 Inc.
 9 216.239.48.154       64.935 ms    6.848 ms    6.793 ms    Google, Inc.
10 72.14.232.141        17.606 ms    17.581 ms    17.680 ms    Google, Inc.
11 209.85.241.27        17.736 ms    17.592 ms    17.519 ms    Google, Inc.
12 72.14.239.193        17.767 ms    17.778 ms    17.930 ms    Google, Inc.
13 74.125.95.106        17.903 ms    17.835 ms    17.867 ms    Google, Inc.:
```

The net is big. It is not uncommon to find a trail of 15 to 20 hops between client and server nowadays. Any machine on any of the subnets the packets travel over can eavesdrop with little difficulty if the packets are not well encrypted.

2.2.1 What OpenSSH Does

The OpenSSH suite gives the following:

- Encrypted remote access, including tunneling insecure protocols.
- Encrypted file transfer
- Run remote commands, programs or scripts and, as mentioned,
- Replacement for **rsh**, **rlogin**, **telnet** and **ftp**

More concretely, that means that the following undesirable activities are prevented:

- Eavesdropping of data transmitted over the network.
- Manipulation of data at intermediate elements in the network (e.g. routers).
- Address spoofing where an attack hosts pretends to be a trusted host by sending packets with the source address of the trusted host.
- IP source routing

As a free software project, OpenSSH provides:

- Open Standards
- Flexible License - freedom emphasized for developers
- Strong Encryption using these ciphers:
  - AES
  - ChaCha20[15]
  - RSA
  - ECDSA
  - Ed25519
- Strong Authentication, supported methods: `gssapi-with-mic, hostbased, keyboard-interactive, none , password` and `publickey`[16]
  - Public Key[10]: can authenticate using multiple keys since March 2015 (OpenSSH 6.8) [17]
  - Single Use Passwords
  - Kerberos
  - Dongles
- Built-in SFTP

10 `https://en.wikibooks.org/wiki/OpenSSH/Cookbook/Public_Key_Authentication`

- Data Compression
- Port Forwarding
  - Encrypt legacy protocols
  - Encrypted X11 forwarding for X Window System
- Key Agents
- Single Sign-on using
  - Authentication Keys
  - Agent Forwarding
  - Ticket Passing
  - Kerberos
  - AFS

2.2.2 What OpenSSH Doesn't Do

OpenSSH is a very useful tool, but much of its effectiveness depends on correct use. It cannot protect from any of the following situations.

- Misconfiguration, misuse, or abuse.
- Compromised systems, particularly where the root account is compromised.
- Insecure or inappropriate directory settings, particularly home directory settings.

OpenSSH must be properly configured and on a properly configured system in order to be of benefit. Arranging both is not difficult, but since each system is unique, there is no one-size-fits-all solution. The right configuration is dependent on the uses the system and OpenSSH are put to.

If you login from a host to a server and an attacker has control of root on either side, he can listen to your session by reading from the pseudo-terminal device because even though SSH is encrypted on the network it must communicate in clear text with the terminal device.

If an attacker can change files in your home directory, for example via a networked file system, he may be able to fool SSH.

Last but not least, if OpenSSH is set to allow everyone in, whether on purpose or by accident, it will.

3 Why Use Encryption

Encryption has been a hot topic in computing for a long time. It became a high priority item in national and international politics in 1991 when Dr. Phil Zimmermann at the Massachusetts Institute of Technology (MIT) first published Pretty Good Privacy (PGP). With the arrival of the first web shops, encryption went from a specialty to a requirement and increasing volumes of money changed hands online. By 1996, encryption became essential for e-business. By 2000, it became recognized as a general, essential prerequisite in electronic communication. Currently, in 2010, there is almost no chance of maintaining control over or integrity of any networked machine for more than a few minutes without the help of encryption.

Nowadays, much communication over computer networks is still done without encryption. That would be most communication, if inadequate encryption is also taken into account. This is despite years of warnings, government recommendations, best practice guidelines and incidents. As a result, any machine connected to the network can intercept communication that passes over that network. The eavesdroppers are many and varied. They include administrators, staff, employers, criminals, corporate spies, and even governments. Corporate espionage alone has become an enormous burden and barrier.

Businesses are well aware of dumpster diving and take precautions to shred all paper documents. But what about electronic information? Contracts and negotiations, trade secrets, patent applications, decisions and minutes, customer data and invoicing, personnel data, financial and tax records, calendars and schedules, product designs and production notes, training materials, and even regular correspondence go over the net daily. Archived materials, even if they are not accessed directly, are usually on machines that are available and accessed for other reasons.

Many company managers and executives are still unaware that their communications and documents are so easily intercepted, in spite of apparent and expensive access restrictions. In many cases these can be shown to be ineffectual and at best purely cosmetic. Security Theater is one obstacle and in the field of security it is more common to find snake oil than authentic solutions. Still, there is little public demonstration of awareness of the magnitude of corporate espionage nowadays or the cost of failure. Even failure to act has its costs. Not only is sensitive data available if left unencrypted, but also trends in less sensitive data can be spotted with a large enough sampling. A very large amount of information can be inferred even from lesser communications. Data mining is now a well-known concept as is the so-called wireless wiretap. With the increase in online material and activity, encryption is more relevant than ever even if many years have passed since the issues were first brought into the limelight.

3.1 Excerpt of ssh-1.0.0 README from July 12, 1995

Tatu Ylönen, then at the Helsinki University of Technology, wrote the README[18] accompanying the early versions of his Open Source software, SSH. The following is an excerpt about why encryption is important.

> ssh-1.0.0 README 1995-07-12
>
> ...
>
> WHY TO USE SECURE SHELL
>
> Currently, almost all communications in computer networks are done without encryption. As a consequence, anyone who has access to any machine connected to the network can listen in on any communication. This is being done by hackers, curious administrators, employers, criminals, industrial spies, and governments. Some networks leak off enough electromagnetic radiation that data may be captured even from a distance.
>
> When you log in, your password goes in the network in plain text. Thus, any listener can then use your account to do any evil he likes. Many incidents have been encountered worldwide where crackers have started programs on workstations without the owners knowledge just to listen to the network and collect passwords. Programs for doing this are available on the Internet, or can be built by a competent programmer in a few days.
>
> Any information that you type or is printed on your screen can be monitored, recorded, and analyzed. For example, an intruder who has penetrated a host connected to a major network can start a program that listens to all data flowing in the network, and whenever it encounters a 16-digit string, it checks if it is a valid credit card number (using the check digit), and saves the number plus any surrounding text (to catch expiration date and holder) in a file. When the intruder has collected a few thousand credit card numbers, he makes smallish mail-order purchases from a few thousand stores around the world, and disappears when the goods arrive but before anyone suspects anything.
>
> Businesses have trade secrets, patent applications in preparation, pricing information, subcontractor information, client data, personnel data, financial information, etc. Currently, anyone with access to the network (any machine on the network) can listen to anything that goes in the network, without any regard to normal access restrictions.
>
> Many companies are not aware that information can so easily be recovered from the network. They trust that their data is safe since nobody is supposed to know that there is sensitive information in the network, or because so much other data is transferred in the network. This is not a safe policy.
>
> Individual persons also have confidential information, such as diaries, love letters, health care documents, information about their personal interests and habits, professional data, job applications, tax reports, political documents, unpublished manuscripts, etc.
>
> There is also another frightening aspect about the poor security of communications. Computer storage and analysis capability has increased so much that it is feasible for governments, major companies, and criminal organizations to automatically analyze,

identify, classify, and file information about millions of people over the years. Because most of the work can be automated, the cost of collecting this information is getting very low.

Government agencies may be able to monitor major communication systems, telephones, fax, computer networks, etc., and passively collect huge amounts of information about all people with any significant position in the society. Most of this information is not sensitive, and many people would say there is no harm in someone getting that information. However, the information starts to get sensitive when someone has enough of it. You may not mind someone knowing what you bought from the shop one random day, but you might not like someone knowing every small thing you have bought in the last ten years.

If the government some day starts to move into a more totalitarian direction, there is considerable danger of an ultimate totalitarian state. With enough information (the automatically collected records of an individual can be manually analyzed when the person becomes interesting), one can form a very detailed picture of the individual's interests, opinions, beliefs, habits, friends, lovers, weaknesses, etc. This information can be used to 1) locate any persons who might oppose the new system 2) use deception to disturb any organizations which might rise against the government 3) eliminate difficult individuals without anyone understanding what happened. Additionally, if the government can monitor communications too effectively, it becomes too easy to locate and eliminate any persons distributing information contrary to the official truth.

Fighting crime and terrorism are often used as grounds for domestic surveillance and restricting encryption. These are good goals, but there is considerable danger that the surveillance data starts to get used for questionable purposes. I find that it is better to tolerate a small amount of crime in the society than to let the society become fully controlled. I am in favor of a fairly strong state, but the state must never get so strong that people become unable to spread contra-official information and unable to overturn the government if it is bad. The danger is that when you notice that the government is too powerful, it is too late. Also, the real power may not be where the official government is.

For these reasons (privacy, protecting trade secrets, and making it more difficult to create a totalitarian state), I think that strong cryptography should be integrated to the tools we use every day. Using it causes no harm (except for those who wish to monitor everything), but not using it can cause huge problems. If the society changes in undesirable ways, then it will be to late to start encrypting.

Encryption has had a ”military” or ”classified” flavor to it. There are no longer any grounds for this. The military can and will use its own encryption; that is no excuse to prevent the civilians from protecting their privacy and secrets. Information on strong encryption is available in every major bookstore, scientific library, and patent office around the world, and strong encryption software is available in every country on the Internet.

Some people would like to make it illegal to use encryption, or to force people to use encryption that governments can break. This approach offers no protection if the government turns bad. Also, the ”bad guys” will be using true strong encryption anyway. Thus, any ”key escrow encryption” or whatever it might be called only serves to help monitor

the ordinary people and petty criminals; it does not help against powerful criminals, terrorists, or espionage, because they will know how to use strong encryption anyway.

...

Thanks also go to Philip Zimmermann, whose PGP software and the associated legal battle provided inspiration, motivation, and many useful techniques, and to Bruce Schneier whose book Applied Cryptography has done a great service in widely distributing knowledge about cryptographic methods.

...

ssh-1.0.0 README 1995-07-12

3.2 Phil Zimmermann on encryption and privacy, from 1991, updated 1999

Phil Zimmermann wrote the encryption tool Pretty Good Privacy (PGP) in 1991 to promote privacy and to help keep encryption, and thus privacy, legal around the world. Considerable difficulty occurred in the United States until PGP was published outside and re-imported in a very visible, public manner.

Why I Wrote PGP
Part of the Original 1991 PGP User's Guide (updated in 1999)

"Whatever you do will be insignificant, but it is very important that you do it."

–Mahatma Gandhi.

It's personal. It's private. And it's no one's business but yours. You may be planning a political campaign, discussing your taxes, or having a secret romance. Or you may be communicating with a political dissident in a repressive country. Whatever it is, you don't want your private electronic mail (email) or confidential documents read by anyone else. There's nothing wrong with asserting your privacy. Privacy is as apple-pie as the Constitution.

The right to privacy is spread implicitly throughout the Bill of Rights. But when the United States Constitution was framed, the Founding Fathers saw no need to explicitly spell out the right to a private conversation. That would have been silly. Two hundred years ago, all conversations were private. If someone else was within earshot, you could just go out behind the barn and have your conversation there. No one could listen in without your knowledge. The right to a private conversation was a natural right, not just in a philosophical sense, but in a law-of-physics sense, given the technology of the time.

But with the coming of the information age, starting with the invention of the telephone, all that has changed. Now most of our conversations are conducted electronically. This allows our most intimate conversations to be exposed without our knowledge. Cellular phone calls may be monitored by anyone with a radio. Electronic mail, sent across the Internet, is no more secure than cellular phone calls. Email is rapidly replacing postal mail, becoming the norm for everyone, not the novelty it was in the past.

Until recently, if the government wanted to violate the privacy of ordinary citizens, they had to expend a certain amount of expense and labor to intercept and steam open and read paper mail. Or they had to listen to and possibly transcribe spoken telephone conversation, at least before automatic voice recognition technology became available. This kind of labor-intensive monitoring was not practical on a large scale. It was only done in important cases when it seemed worthwhile. This is like catching one fish at a time, with a hook and line. Today, email can be routinely and automatically scanned for interesting keywords, on a vast scale, without detection. This is like driftnet fishing. And exponential growth in computer power is making the same thing possible with voice traffic.

Perhaps you think your email is legitimate enough that encryption is unwarranted. If you really are a law-abiding citizen with nothing to hide, then why don't you always send your paper mail on postcards? Why not submit to drug testing on demand? Why require a warrant for police searches of your house? Are you trying to hide something? If you hide your mail inside envelopes, does that mean you must be a subversive or a drug dealer, or maybe a paranoid nut? Do law-abiding citizens have any need to encrypt their email?

What if everyone believed that law-abiding citizens should use postcards for their mail? If a nonconformist tried to assert his privacy by using an envelope for his mail, it would draw suspicion. Perhaps the authorities would open his mail to see what he's hiding. Fortunately, we don't live in that kind of world, because everyone protects most of their mail with envelopes. So no one draws suspicion by asserting their privacy with an envelope. There's safety in numbers. Analogously, it would be nice if everyone routinely used encryption for all their email, innocent or not, so that no one drew suspicion by asserting their email privacy with encryption. Think of it as a form of solidarity.

Senate Bill 266, a 1991 omnibus anticrime bill, had an unsettling measure buried in it. If this non-binding resolution had become real law, it would have forced manufacturers of secure communications equipment to insert special ”trap doors” in their products, so that the government could read anyone's encrypted messages. It reads, ”It is the sense of Congress that providers of electronic communications services and manufacturers of electronic communications service equipment shall ensure that communications systems permit the government to obtain the plain text contents of voice, data, and other communications when appropriately authorized by law.” It was this bill that led me to publish PGP electronically for free that year, shortly before the measure was defeated after vigorous protest by civil libertarians and industry groups.

The 1994 Communications Assistance for Law Enforcement Act (CALEA) mandated that phone companies install remote wiretapping ports into their central office digital switches, creating a new technology infrastructure for ”point-and-click” wiretapping, so that federal agents no longer have to go out and attach alligator clips to phone lines. Now they will be able to sit in their headquarters in Washington and listen in on your phone calls. Of course, the law still requires a court order for a wiretap. But while technology infrastructures can persist for generations, laws and policies can change overnight. Once a communications infrastructure optimized for surveillance becomes entrenched, a shift in political conditions may lead to abuse of this new-found power. Political conditions may shift with the election of a new government, or perhaps more abruptly from the bombing of a federal building.

A year after the CALEA passed, the FBI disclosed plans to require the phone companies to build into their infrastructure the capacity to simultaneously wiretap 1 percent of all phone calls in all major U.S. cities. This would represent more than a thousandfold increase over previous levels in the number of phones that could be wiretapped. In previous years, there were only about a thousand court-ordered wiretaps in the United States per year, at the federal, state, and local levels combined. It's hard to see how the government could even employ enough judges to sign enough wiretap orders to wiretap 1 percent of all our phone calls, much less hire enough federal agents to sit and listen to all that traffic in real time. The only plausible way of processing that amount of traffic is a massive Orwellian application of automated voice recognition technology to sift through it all, searching for interesting keywords or searching for a particular speaker's voice. If the government doesn't find the target in the first 1 percent sample, the wiretaps can be shifted over to a different 1 percent until the target is found, or until everyone's phone line has been checked for subversive traffic. The FBI said they need this capacity to plan for the future. This plan sparked such outrage that it was defeated in Congress. But the mere fact that the FBI even asked for these broad powers is revealing of their agenda.

Advances in technology will not permit the maintenance of the status quo, as far as privacy is concerned. The status quo is unstable. If we do nothing, new technologies will give the government new automatic surveillance capabilities that Stalin could never have dreamed of. The only way to hold the line on privacy in the information age is strong cryptography.

You don't have to distrust the government to want to use cryptography. Your business can be wiretapped by business rivals, organized crime, or foreign governments. Several foreign governments, for example, admit to using their signals intelligence against companies from other countries to give their own corporations a competitive edge. Ironically, the United States government's restrictions on cryptography in the 1990's have weakened U.S. corporate defenses against foreign intelligence and organized crime.

The government knows what a pivotal role cryptography is destined to play in the power relationship with its people. In April 1993, the Clinton administration unveiled a bold new encryption policy initiative, which had been under development at the National Security Agency (NSA) since the start of the Bush administration. The centerpiece of this initiative was a government-built encryption device, called the Clipper chip, containing a new classified NSA encryption algorithm. The government tried to encourage private industry to design it into all their secure communication products, such as secure phones, secure faxes, and so on. AT&T put Clipper into its secure voice products. The catch: At the time of manufacture, each Clipper chip is loaded with its own unique key, and the government gets to keep a copy, placed in escrow. Not to worry, though–the government promises that they will use these keys to read your traffic only ”when duly authorized by law.” Of course, to make Clipper completely effective, the next logical step would be to outlaw other forms of cryptography.

The government initially claimed that using Clipper would be voluntary, that no one would be forced to use it instead of other types of cryptography. But the public reaction against the Clipper chip was strong, stronger than the government anticipated. The computer industry monolithically proclaimed its opposition to using Clipper. FBI director Louis Freeh responded to a question in a press conference in 1994 by saying

that if Clipper failed to gain public support, and FBI wiretaps were shut out by non-government-controlled cryptography, his office would have no choice but to seek legislative relief. Later, in the aftermath of the Oklahoma City tragedy, Mr. Freeh testified before the Senate Judiciary Committee that public availability of strong cryptography must be curtailed by the government (although no one had suggested that cryptography was used by the bombers).

The government has a track record that does not inspire confidence that they will never abuse our civil liberties. The FBI's COINTELPRO program targeted groups that opposed government policies. They spied on the antiwar movement and the civil rights movement. They wiretapped the phone of Martin Luther King. Nixon had his enemies list. Then there was the Watergate mess. More recently, Congress has either attempted to or succeeded in passing laws curtailing our civil liberties on the Internet. Some elements of the Clinton White House collected confidential FBI files on Republican civil servants, conceivably for political exploitation. And some overzealous prosecutors have shown a willingness to go to the ends of the Earth in pursuit of exposing sexual indiscretions of political enemies. At no time in the past century has public distrust of the government been so broadly distributed across the political spectrum, as it is today.

Throughout the 1990s, I figured that if we want to resist this unsettling trend in the government to outlaw cryptography, one measure we can apply is to use cryptography as much as we can now while it's still legal. When use of strong cryptography becomes popular, it's harder for the government to criminalize it. Therefore, using PGP is good for preserving democracy. If privacy is outlawed, only outlaws will have privacy.

It appears that the deployment of PGP must have worked, along with years of steady public outcry and industry pressure to relax the export controls. In the closing months of 1999, the Clinton administration announced a radical shift in export policy for crypto technology. They essentially threw out the whole export control regime. Now, we are finally able to export strong cryptography, with no upper limits on strength. It has been a long struggle, but we have finally won, at least on the export control front in the US. Now we must continue our efforts to deploy strong crypto, to blunt the effects increasing surveillance efforts on the Internet by various governments. And we still need to entrench our right to use it domestically over the objections of the FBI.

PGP empowers people to take their privacy into their own hands. There has been a growing social need for it. That's why I wrote it.

Philip R. Zimmermann
Boulder, Colorado

June 1991 (updated 1999)[6]

3.3 Original Press Release for OpenSSH

Below is the original press release for OpenSSH sent back in 1999.[19]

Date: Mon, 25 Oct 1999 00:04:29 -0600 (MDT)

From: Louis Bertrand <louiscvs.openbsd.org>

To: Liz Coolbaugh <lwnlwn.net>

Subject: OpenBSD Press Release: OpenSSH integrated into operating system

PRESS RELEASE

OpenSSH: Secure Shell integrated into OpenBSD operating system

Source: OpenBSD

Contacts:

Louis Bertrand, OpenBSD media relations

Bertrand Technical Services

Tel: (905) 623-8925 Fax: (905) 623-3852

louisopenbsd.org

Theo de Raadt, OpenBSD lead developer

deraadtopenbsd.org

Project Web site: [1]

OpenSSH: Secure Shell integrated into OpenBSD Secure communications package no longer third-party add-on

[October 25, 1999: Calgary, Canada] – The OpenBSD developers are pleased to announce the release of OpenSSH, a free implementation of the popular Secure Shell encrypted communications package. OpenSSH, to be released with OpenBSD 2.6, is compatible with both SSH 1.3 and 1.5 protocols and dodges most restrictions on the free distribution of strong cryptography.

OpenSSH is based on a free release of SSH by Tatu Ylonen, with major changes to remove proprietary code and bring it up to current security and functionality standards. Secure Shell operates like the popular TELNET remote terminal package but with an encrypted link between the user and the remote server. SSH also allows ”tunnelling” of network services through the scrambled connection for added privacy. OpenSSH has been tested to interoperate with ssh-1.2.27 from SSH Communications, and the TTSSH and SecureCRT Windows clients.

”Network sessions involving strong cryptographic security are a requirement in the modern world,” says lead developer Theo de Raadt. ”Everyone needs this. People using the

1 `http://www.openbsd.org/`

telnet or rlogin protocols are not aware of the extreme danger posed by password sniffing and session hijacking."

In previous releases of OpenBSD, users were urged to download SSH as soon as possible after installing the OS. Without SSH, terminal sessions transmitted in clear text allow eavesdroppers on the Internet to capture user names and password combinations and thus bypass the security measures in the operating system.

"I asked everyone 'what is the first thing you do after installing OpenBSD?' Everyone gave me the same answer: they installed ssh," says de Raadt. "That's a pain, so we've made it much easier."

All proprietary code in the original distribution was replaced, along with some libraries burdened with the restrictive GNU Public License (GPL). Much of of the actual cryptographic code was replaced by calls to the crypto libraries built into OpenBSD. The source code is now completely freely re-useable, and vendors are encouraged to re-use it if they need ssh functionality.

OpenSSH relies on the Secure Sockets Layer library (libssl) which incorporates the RSA public-key cryptography system. RSA is patented in the US and OpenBSD developers must work around the patent restrictions. Users outside the US may download a libssl file based on the patent-free OpenSSL implementation. For US non-commercial users, OpenBSD is preparing a libssl based on the patented RSAREF code. Unfortunately, the US legal framework effectively bans US commercial users from using OpenSSH, and curtails freedom of choice in that market.

OpenSSH was developed and integrated into OpenBSD by Niels Provos, Theo de Raadt, Markus Friedl for cryptographic work; Dug Song, Aaron Campbell, and others for various non-crypto contributions; and Bob Beck for helping with the openssl library issues. The original SSH was written by Tatu Ylonen. Bjoern Groenvall and Holger Trapp did the initial work to free the distribution.

OpenBSD is an Internet-based volunteer effort to produce a secure multi-platform operating system with built-in support for cryptography. It has been described in the press as the world's most secure operating system. For more information about OpenSSH and OpenBSD, see the project Web pages at [2].

Source: OpenBSD

[3]

3.4 The European Union (EU) on Encryption

During 2000, the European Commission investigated the state of international and industrial electronic espionage. Counter-measures and solutions were investigated as well as the risks. The result was a resolution containing a summary of the findings and a series of recommended actions for Member States to carry out and goals to meet. Recommendations

2 http://www.OpenBSD.org/

3 http://lwn.net/1999/1028/a/openssh.html

to EU Member States from the European Parliament resolution ECHELON, A5-0264/2001 (emphasis added):

> ”29. Urges the Commission and Member States to devise appropriate measures to promote, develop and manufacture European encryption technology and software and above all to support projects at developing user-friendly **open-source encryption** software;”
>
> . . .
>
> ”33. Calls on the Community institutions and the public administrations of the Member States to provide training for their staff and make their staff familiar with new **encryption technologies and techniques** by means of the necessary practical training and courses;” [20]

It was found during the investigation that businesses were the most at risk and the most vulnerable and that widespread use of open source encryption technology is to be encouraged. The same can be said even today.

4 SSH Protocols

OpenSSH uses the SSH protocol which connects over TCP. Normally, one SSH session per TCP connection is made, but multiple sessions can be multiplexed over a single TCP connection if planned that way. The current set of Secure Shell protocols is SSH2. It is a rewrite of the old, deprecated SSH1 protocol. It contains significant improvements in security, performance, and portability. The default is now SSH2 and SSH1 support has been removed from both the client and server.

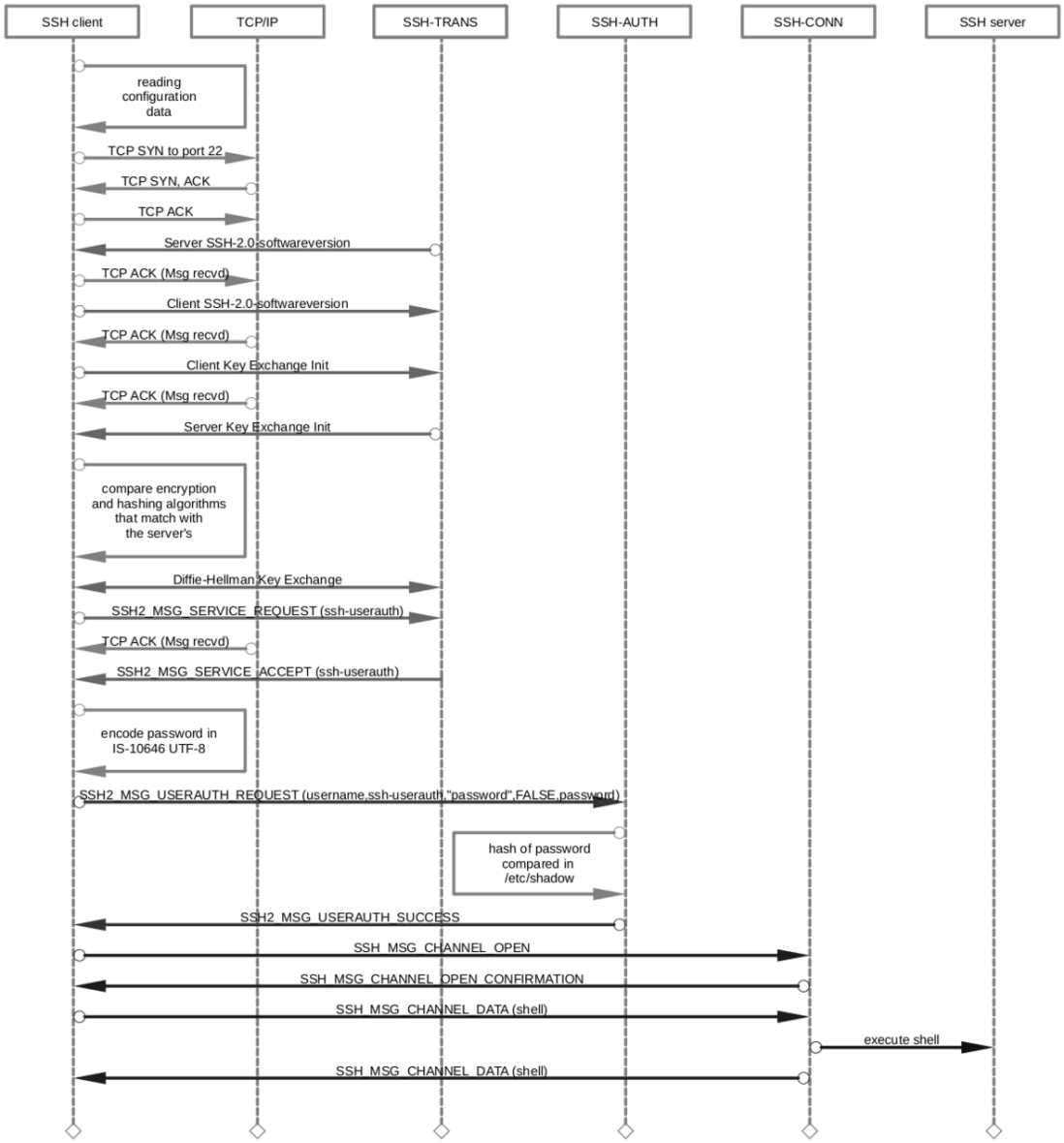

Figure 2 Sequence Diagram for SSH Password Authentication

The Secure Shell protocol is an open standard. As such, it is vendor-neutral and maintained by the Internet Engineering Task Force (IETF). The current protocol is described in RFC 4250[1] through RFC 4256[2] and standardized by the IETF secsh working group. The overall structure of SSH2 is described in RFC 4251[3], The Secure Shell (SSH) Protocol Architecture.

The SSH protocol is composed of three layers: the transport layer, the authentication layer, and the connection layer.

1 `https://tools.ietf.org/html/rfc4250`
2 `https://tools.ietf.org/html/rfc4256`
3 `https://tools.ietf.org/html/rfc4251`

SSH-CONNECT – The connection layer runs over the user authentication protocol. It multiplexes many different concurrent encrypted channels into logical channels over the authenticated connection. It allows for tunneling of login sessions and TCP-forwarding. It provides a flow control service for these channels. Additionally, various channel-specific options can be negotiated. This layer manages the SSH session, session multiplexing, X11 forwarding, TCP forwarding, shell, remote program execution, invoking SFTP subsystem.

SSH-USERAUTH – The user authentication layer authenticates the client-side to the server. It uses the established connection and runs on top of the transport layer. It provides several mechanisms for user authentication. These include password authentication, public-key or host-based authentication mechanisms, challenge-response, pluggable authentication modules (PAM), Generic Security Services API (GSSAPI) and even dongles.

SSH-TRANS – The transport layer provides server authentication, confidentiality and data integrity over TCP. It does this through algorithm negotiation and a key exchange. The key exchange includes server authentication and results in a cryptographically secured connection: it provides integrity, confidentiality and optional compression. [21]

Among the differences between the current protocol, SSH2, and the deprecated SSH1 protocol, is that SSH2 uses host keys for authentication. Whereas SSH1 used both server and host keys to authenticate. There's not much which can be added about the protocols which is not already covered with more detail and authority in RFC 4251[4] [22].

4.1 SSH File Transfer Protocol (SFTP)

The SSH File Transfer Protocol (SFTP) is a binary protocol to provide secure file transfer, access and management.

SFTP was added by Markus Friedl on the server side in time for the 2.3.0 release of OpenSSH in November 2000. Damien Miller added support for SFTP to the client side in time for 2.5.0. Since then, many have added to both the client and the server.

4.1.1 SFTP is not FTPS

For basic file transfer, nothing more is needed than an account on the machine with the OpenSSH server. SFTP support is built into the OpenSSH server package. The SFTP protocol, in contrast to old FTP, has been designed from the ground up to be as secure as possible for both login and data transfer.

Samba[5] is an interoperability suite providing fast file and print services for any client which can use the SMB/CIFS protocol. It is most often used for file sharing at the level of the local area network. **AFS**[6], or the Andrew File System, is a distributed file system providing file sharing at an institutional level across a geographically diverse institution or set of collaborating institutions. Notably it provides a set of trusted servers and a homogeneous,

4 https://tools.ietf.org/html/rfc4251
5 https://en.wikipedia.org/wiki/Samba_(software)
6 https://en.wikipedia.org/wiki/Andrew_File_System

location-transparent file name space to all the client workstations. Unless the use-case calls for publicly available, read-only, downloads, don't worry about trying to fiddle with FTP. It is the protocol FTP itself that is inherently insecure. It's great for read-only, public data transfer. The programs **vsftpd** and **proftpd**, for example, are secure insofar as the server software itself goes, although the protocol itself is still insecure. In other words the program itself is more or less fine and if you need to provide read-only, publicly available downloads then FTP maybe the right tool. Otherwise forget about FTP. Nearly always when users ask for "FTP" they don't mean specifically the old file transfer protocol from 1971 as described in RFC 114[7], but a generic means of file transfer and there are many ways to solve that problem. This is especially true since the next part of their request is usually how to make it secure. The name "FTP" is frequently mis-used generically to mean any file transfer utility, much the same way as the term "Coke" is used in some of Southern United States to mean any carbonated soft drink, not just Coca-Cola. Consider SFTP or, for larger groups, even SSHFS, Samba[8], or AFS[9]. While old FTP succeeded very well in achieving its main goal to promote use of networked computers by allowing users at any host on the network to use the file system of any cooperating host, it cannot be made secure. There's nothing to be done about that, so it is past time to get over it.

Again, it is the protocol itself, FTP, which is the problem.[23] With FTP, the data, passwords and user name are all sent back and forth unencrypted.[24] Anyone on the client's subnet, the server's subnet or any subnet in between can 'sniff' the passwords and data when FTP is used. With extra effort it is possible to wrap FTP inside SSL or TLS, thus creating FTPS. However, tunneling FTP over SSL/TLS is complex to do and far from an optimum solution.

Unfortunately because of name confusion combined with the large number of posts and discussions created by complex, nit-picky tasks like wrapping FTP in SSL to provide FTPS, the wrong way still turns up commonly in web searches regarding file transfer. In contrast, easy, relatively painless solutions vanish because it is rarely necessary to post how to do those. Also, an easy solution can be summed up in very few lines and maybe a single answer. Thus, there is still a lot of talk online about 'securing' FTP and very little mention of using SFTP. It's a vicious cycle that this book hopes to help break: Difficult tasks mean lots of discussion and noise, lots of discussion and noise means strong web presence, strong web presence means high Google ranking.

SFTP tools are very common, but might be taken for granted and thus overlooked. SFTP tools are easy to use and more functional than old FTP clients. In fact a lot of improvements have been made in usability. There is no shortage of common, GUI-based SFTP clients to transfer files: Filezilla, Konqueror, Dolphin, Nautilus, Cyberduck, Fugu, and Fetch top the list but there are many more. Most are Free Software. Again, these SFTP clients are very easy to use. For example, in Konqueror, just type in the URL to the sftp server, where the server name or address is xx.yy.zz.aa.

```
sftp://xx.yy.zz.aa
```

If it is desirable to start with a particular directory, then that too can be specified.

7 https://tools.ietf.org/html/rfc114
8 https://en.wikipedia.org/wiki/Samba_(software)
9 https://en.wikipedia.org/wiki/Andrew_File_System

```
sftp://xx.yy.zz.aa/var/www/pictures/
```

One special client worth knowing about is **sshfs**. With **sshfs** as an SFTP client the other machine is accessible as an open folder on your machine's local file system. In that way any program you normally have to work with files, such as LibreOffice, Inkscape or Gimp can access the remote machine via that folder.

4.1.2 Background of FTP

FTP is from the 1970s. It's a well proven workhorse, but from an era when if you were on the net you were supposed to be there and if there was trouble it could usually be cleared up with a short phone call or an e-mail or two. It sends the login name, password and all of the data unencrypted for anyone to intercept. FTP clients can connect to the FTP server in either passive or active modes. Both active and passive modes for FTP[25] use two ports, one for control and one for data. In FTP Active mode, after the client makes a connection to the FTP server it then allows an incoming connection to be initiated from the server to for data transfer. In FTP Passive mode, after the client makes a connection to the FTP server, the server then responds with information about a second port for data transfer and the client initiates the second connection. FTP is most relevant now as Anonymous FTP, which is still excellent for read-only downloads without login. FTP is still one way to go for transferring read-only data, as would be using the web (HTTP or HTTPS), or a P2P protocol like Bittorrent. So there are other options than FTP for offering read-only downloads. Preference is given lately to HTTPS for small files and Bittorrent for large files or large groups of files.

Using tcpdump to show FTP activity

An illustration of how the old protocol, FTP, is insecure can be had from the utility **tcpdump**. It can show what is going over the network during an Anonymous FTP session, or for that matter any FTP session. Look at the manual page for **tcpdump** for an explanation of the individual arguments, the usage example below displays the first FTP or FTP-Data packets going from the client to the server and vice versa.

The output below shows an excerpt from the output of **tcpdump** which captured packets between an FTP client and the FTP server, one line per packet.

```
$ sudo tcpdump -q -s 0 -c 10 -A -i eth0 \
"tcp and (port ftp or port ftp-data)"
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 65535 bytes
…
:18:36.010820 IP desk.55227 > server.ftp: tcp 16 E..D..@.@.....1.[.X.....G.r.
 ........l..... ."......USER anonymous
:18:36.073192 IP server.ftp > desk.55227: tcp 0 E..4jX@.7.3.[.X...1.....
 ...G.r#.......... ....."..
:18:36.074019 IP server.ftp > desk.55227: tcp 34 E..VjY@.7.3.[.X...1.....
 ...G.r#....Y...... ....."..331 Please specify the password.
:18:36.074042 IP desk.55227 > server.ftp: tcp 0 E..4..@.@..+..1.[.X.....G.r#
 ..)........... ."......
:18:42.098941 IP desk.55227 > server.ftp: tcp 23 E..K..@.@.....1.[.X.....G.r#
 ..)....gv..... .".w....PASS user@example.net
:18:42.162692 IP server.ftp > desk.55227: tcp 23 E..KjZ@.7.3.[.X...1.....
```

```
 ..)G.r:.......... ....."..w230 Login successful.
…
:18:43.431827 IP server.ftp > desk.55227: tcp 14 E..Bj\@.7.3.[.X...1.....
 ..SG.rF.....j..... ....."..221 Goodbye.
…
```

As can be seen in lines 3 and 7, data such as text from the server is visible. In lines 1 and 5, text entered by the user is visible and in this case it includes the user name and password used to log in. Fortunately the session is Anonymous FTP, which is read-only and used for downloading. Anonymous FTP is a rather efficient way to publish material for download. For Anonymous FTP, the user name is always ”anonymous” and the password is the user’s e-mail address and the server's data always read-only.

If you have the package for the OpenSSH server already installed, no further configuration of the server is needed to start using SFTP for file transfers. Though comparatively speaking, FTPS is significantly more secure than FTP. If you want remote remote login access, then both FTP and FTPS should be avoided. A very large reason to avoid both is to save work.

4.1.3 On FTPS

FTPS is FTP tunneled over SSL or TLS. A goal of FTP was to encourage the use of remote computers which, along with the web, has succeeded. A goal of FTPS was to secure logins and transfers, and it was a necessary step in securing file transfers with the legacy protocol. However, since SFTP is so much easier to deploy and most systems now include both graphical and text-based SFTP clients, FTPS can really be considered deprecated for most occasions.

Some good background material can be found in the Request for Comments (RFCs) for FTP and FTPS. There, SFTP and even HTTPS are better matches and largely supersede FTPS. See the section on Client Applications[10] for an idea of the SFTP clients available.

10 https://en.wikibooks.org/wiki/OpenSSH/Client_Applications

4.2 Privilege Separation

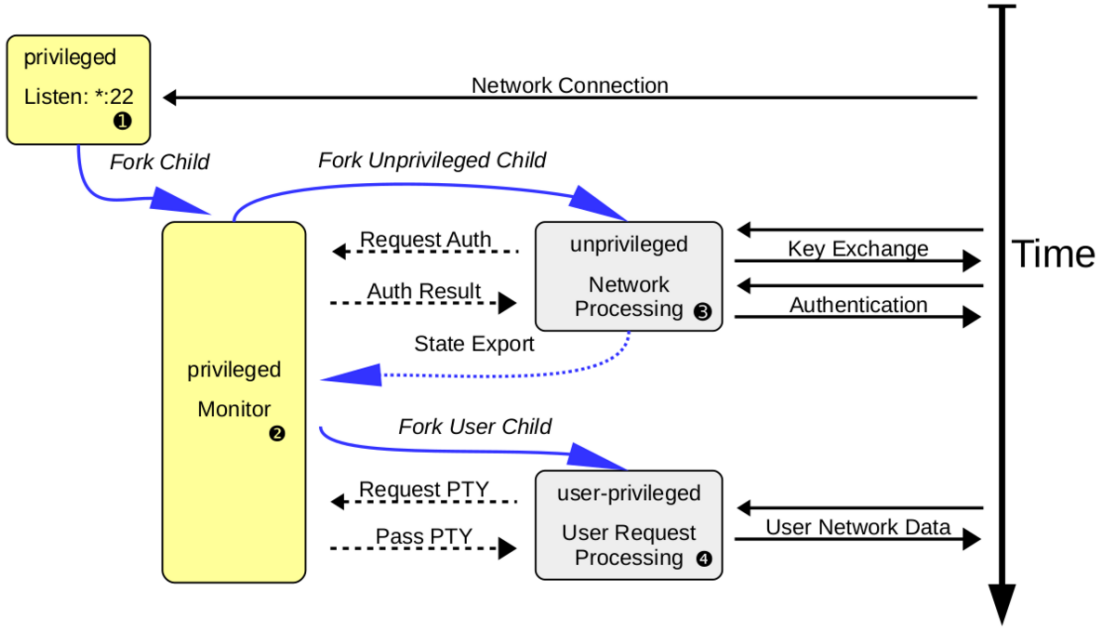

Figure 3 Sequence Diagram for OpenSSH Privilege Separation

Privilege separation is when a process is divided into sub-processes, each of which have just enough access to just the right services to do their part of the job. An underlying principle is that of least privilege, which is where each process has exactly enough privileges to accomplish a task, neither more nor less. The goal of privilege separation is to compartmentalize any corruption and prevent a corrupt process from accessing other parts of the system. Privilege separation is applied in OpenSSH by using several levels of access, some higher some lower, to run sshd(8)[11] and its subsystems and components. The SSH server ❶ starts out with a privileged process ❷which then creates an unprivileged process ❸to work with the network traffic. Once the user has authenticated, another unprivileged process is created ❹with the privileges of that authenticated user. See the "Sequence Diagram for OpenSSH Privilege Separation". As seen in the diagram, a total of four processes get run to create an SSH session. One process, the server, remains and listens for new connections and spawn new child processes.

```
$ ps -ax -o user,pid,ppid,state,start,command | awk '/sshd/ || NR==1'
USER       PID  PPID STAT   STARTED  COMMAND
root      1473     1 I     05:44:01  sshd: /usr/sbin/sshd [listener] 0 of 10-10
```

It is this privileged process that listens for the initial connection from clients. Here it is seen waiting and listening on port 22.

```
$ netstat -ntlp | awk '/sshd/ || NR<=2'
Active Internet connections (only servers)
```

11 http://man.openbsd.org/sshd.8

```
Proto Recv-Q Send-Q Local Address           Foreign Address         State
 PID/Program name
tcp        0      0 0.0.0.0:22              0.0.0.0:*               LISTEN
 1473/sshd
tcp6       0      0 :::22                   :::*                    LISTEN
 1473/sshd
```

After the initial connection while waiting for password authentication from user 'fred', a privileged monitor process supervises an unprivileged process by user 'sshd' which handles the contact with the remote user's client.

```
$ ps -ax -o user,pid,ppid,state,start,command | awk '/sshd/ || NR==1'
USER       PID  PPID S  STARTED COMMAND
root      1473     1 S 05:44:12 sshd: /usr/sbin/sshd [listener] 1 of 10-10
root      9481  1473 S 14:40:37 sshd: fred [priv]
sshd      9482  9481 S 14:40:37 sshd: fred [net]
```

Then after authentication is completed and a session established for user 'fred', a new privileged monitor process is created to supervise the process running as user 'fred'. At that point the other process running as user 'sshd' has gone away.

```
$ ps -ax -o user,pid,ppid,state,start,command | awk '/sshd/ || NR==1'
USER       PID  PPID S  STARTED COMMAND
root      1473     1 S 05:44:12 sshd: /usr/sbin/sshd [listener] 0 of 10-10
root      9481  1473 S 14:40:37 sshd: fred [priv]
fred      9579  9481 S 14:42:02 sshd: fred@pts/30
```

Privilege separation has been the default in OpenSSH since version 3.3[26] Since version 5.9, privilege separation further applies mandatory restrictions on which system calls the privilege separated child can perform. The intent is to prevent a compromised privilege separated child from being used to attack other hosts either by opening sockets and proxying or by probing local kernel attack surface. [27] Since version 6.1, this sandboxing has been the default.

5 Other SSH Implementations

5.1 Dropbear

Dropbear[1] is a smaller, modular, open source SSH2 client and server available for all regular POSIX platforms. Dropbear is partially a derivative of OpenSSH and it is often used in embedded systems because very small binaries can be produced. Functions that are not needed can be left out of the binary, leaving a lean executable. Thus a working SSH server can be boiled down to 110KB by trimming away various functions.

Many distributions and products use Dropbear. This includes OpenWRT, gumstix, Tomato Firmware, PSPSSH, DSLinux, Meego, OpenMoko, Ångström (for Zaurus), ttylinux, Sisela, Trinux, SliTaz, Netcomm, US Robotics, some Motorola phones, and many, many more.

- [2]

5.2 Tectia

Tectia[3] is from SSH Communications Security Corporation which is based in Finland. It is a closed-source SSH client and server with FIPS support.

- [4]

5.3 Solaris Secure Shell (SunSSH)

Sun SSH[5] is fork of OpenSSH 2.3, with many subsequent changes.

- [6]

5.4 GlobalSCAPE EFT Server

EFT Server[7] is a closed binary that can include SSH and SFTP modules as extensions.

- [8]

1 https://matt.ucc.asn.au/dropbear/dropbear.html
2 https://matt.ucc.asn.au/dropbear/dropbear.html
3 http://www.tectia.com/en.iw3
4 http://www.ssh.com/?option=com_content&view=article&id=236&Itemid=364
5 http://wikis.sun.com/display/SunSSH/SunSSH+FAQ
6 http://hub.opensolaris.org/bin/view/Community+Group+security/SSH
7 http://www.globalscape.com/eft/
8 http://www.globalscape.com/eft/

5.5 Gravitational Teleport

Teleport[9] provides an Apache-licensed SSH server and client written in Golang. It supports only IPv4 and not IPv6 at this time.

- [10]

9 https://gravitational.com/teleport/
10 https://gravitational.com/teleport/

6 Client Applications

On the client side, ssh(1)[1], scp(1)[2], and sftp(1)[3] provide a wide range of capabilities. Interactive logins and file transfers are just the tip of the iceberg.

ssh(1)[4] - The basic login shell-like client program.

sftp(1)[5] - FTP-like program that works using the SSH protocol.

scp(1)[6] - File copy program that acts like rcp(1).

ssh_config(5)[7] - The client configuration file.

6.1 The SSH client

ssh(1)[8] is a program which provides the client side for secure, encrypted communications between hosts over an insecure network. Its main use is for logging into and running programs on a remote host. It can also be used to secure remote X11 connections and forward arbitrary TCP ports to secure legacy protocols. ssh was made, in part, to replace insecure tools like **rsh** and **telnet**. It has largely succeeded at this goal. **rsh** and **telnet** are rarely seen anymore for interactive sessions or anywhere else. ssh can authenticate using regular passwords or with the help of a public-private key pair. More options, such as use of Kerberos, smartcards, or one-time passwords can be configured.

Remote login, authenticating via password:

```
$ ssh fred@somehost.example.org
```

Another way of logging in to the same account:

```
$ ssh -l fred somehost.example.org
```

Remote programs can be run interactively when the client is run via the shell on the remote host. Or they can be run directly when passed as an argument to the SSH client. They can even be pre-configured in the authentication key or the server configuration.

1 http://man.openbsd.org/ssh.1
2 http://man.openbsd.org/scp.1
3 http://man.openbsd.org/sftp.1
4 http://man.openbsd.org/ssh.1
5 http://man.openbsd.org/sftp.1
6 http://man.openbsd.org/scp.1
7 http://man.openbsd.org/ssh_config.5
8 http://man.openbsd.org/ssh.1

Run uname(1)[9] on the remote machine:

```
$ ssh -l fred  somehost.example.org  "uname -a"
```

See what file systems are mounted and how much space is used there:

```
$ ssh -l fred  somehost.example.org  "mount; df -h"
```

It is possible to configure in great detail which programs are allowed by which accounts. There are many combinations of options that give extra capabilities, such as re-using a single connection for multiple sessions or passing through intermediary machines. The level of granularity can be increased even more with the help of **sudo(8)**.

6.1.1 SSH Client Environment Variables – Server Side

Of course the foundation of most SSH activity centers around use of the shell. Upon a successful connection, OpenSSH sets several environment variables.

```
SSH_CLIENT='192.168.223.17 36673 22'
SSH_CONNECTION='192.168.223.17 36673 192.168.223.229 22'
SSH_TTY=/dev/pts/6
```

SSH_CLIENT shows the address of the client system, the outgoing port number on the client system, and the incoming port on the server. **SSH_CONNECTION** shows the address of the client, the outgoing port on the client, the address of the server and the incoming port on the server. **SSH_TTY** names the pseudo-terminal device, abbreviated PTY, on the server used by the connection. For more information on pseudo-terminals see ptm(4)[10], tty(1)[11] and tty(4)[12].

The login session can be constrained to a single program with a predetermined set of parameters using **ForceCommand** in the server configuration or **Command="..."** in the authorized keys file. When that happens an additional environment variable **SSH_ORIGINAL_COMMAND** gets set.

```
SSH_ORIGINAL_COMMAND=echo "hello, world"
```

If the server has **ExposeAuthInfo** set, then the **SSH_USER_AUTH** environment variable points to a temporary file listing details about the authentication methods used to start the current session.

```
SSH_USER_AUTH=/tmp/sshauth.4JmbYfF0bhF6C17
```

The file is removed when the session ends.

Other variables are set depending on the user's shell settings and the system's own settings.

9 http://man.openbsd.org/uname.1
10 http://man.openbsd.org/ptm.4
11 http://man.openbsd.org/tty.1
12 http://man.openbsd.org/tty.4

6.1.2 SSH Client Configuration Options

GSSAPI[13], or Generic Security Service Application Program Interface, a standard interface defined by RFC 2743[14] to provide a means for independent modules with means for generic authentication and secure messaging. Kerberos V is one of the more common examples of a service using it. Configuration options can be passed to ssh(1)[15] as arguments, see the manual pages for ssh(1)[16] and ssh_config(5)[17] for the full list.

Connect very verbose output, GSSAPI[18] authentication:

```
$ ssh -vv -K -l account host.example.org
```

A subset of options can be defined on the server host in the user's own authorized keys file, in conjunction with specific keys. See sshd(8)[19] for which subset exactly.

```
command="/usr/local/sbin/backup.sh",no-pty ssh-rsa
 AAAAB3NzaC1yc2EAAAQEAsY6u71N...
command="/usr/games/wump",no-port-forwarding,no-pty ssh-ed25519
 AAAAC3NzaC11ZDI1...
environment="gt
m_dist=/usr/local/gtm/utf8",environment="gtm_principal_editing=NOINSERT:EDITING"
 ssh-rsa AAAA8a2s809poloh05yhh...
```

Note that some directives, like setting the environment variables, are disabled by default and must be named in the server configuration before available to the client. More configuration directives can be set by the user in **~/.ssh/config** or by the system administrator in **/etc/ssh/ssh_config**. These same configuration directives can be passed as arguments using **-o**. See ssh_config(5)[20] for the full list with descriptions.

```
$ ssh -o "ServerAliveInterval=60" -o "Compression=yes" -l fred
 server.example.org
```

The system administrators of the client host can set some global defaults in **/etc/ssh/config**. Some of these global settings can be targeted to a specific group or user by using a **Match** directive.

For example, if a particular SSH server is available via port 2022, it may be convenient to have the client try that port automatically. Some of OpenBSD's anonymous CVS servers accept SSH connections on this port. However, compression should not be used in this case because CVS already uses compression. So it should be turned off. So, one could specify something like the following in the **$HOME/.ssh/config** configuration file so that the default port is 2022 and the connection is made without compression:

13 https://en.wikipedia.org/wiki/Generic_Security_Services_Application_Program_Interface
14 https://tools.ietf.org/html/rfc2743
15 http://man.openbsd.org/ssh.1
16 http://man.openbsd.org/ssh.1
17 http://man.openbsd.org/ssh_config.5
18 https://en.wikipedia.org/wiki/Generic_Security_Services_Application_Program_Interface
19 http://man.openbsd.org/sshd.8
20 http://man.openbsd.org/ssh_config.5

```
Host anoncvs anoncvs.example.org
        Compression no
        Port 2022
```

See ssh_config(5)[21] for the client side and sshd_config(5)[22] for the server side for the full lists with descriptions.

6.2 The SFTP client

sftp(1)[23] is an interactive file transfer program which performs all its operations over an encrypted SSH transport channel. It may also use many features of ssh(1)[24], such as public key authentication and compression. It is also the name of the protocol used.

The SFTP protocol is similar in some ways to the now venerable File Transfer Protocol (FTP), except that the entire session, including the login, is encrypted. However, SFTP is not FTPS. The latter old-fashioned FTP tunneled over SSH/SSL. In contrast, SFTP is actually a whole new protocol. sftp(1)[25] can also be made to start in a specific directory on the remote host.

```
$ sftp fred@server.example.org:/var/www
```

Frequently, SFTP is used to connect and log into a specified host and enter an interactive command mode. See the manual page for sftp(1)[26] for the available interactive commands such as **get**, **put**, **rename**, and so on. Also, the same configuration options that work for ssh(1)[27] also apply to sftp(1)[28]. sftp(1)[29] accepts all ssh_config(5)[30] options and these can be passed along as arguments at run time. Some have explicit shortcuts.

```
$ sftp -i ~/.ssh/some.key.ed25519 fred@server.example.org:/var/www
```

While others can be specified by naming them in full using the **-o** option.

```
$ sftp -o "ServerAliveInterval=60" -o "Compression=yes" fred@server.example.org
```

Another way to transfer is to send or receive files automatically. If a non-interactive authentication method is used, the whole process can be automatic using batch mode.

```
$ sftp -b session.batch -i ~/.ssh/some_key_rsa fred@server.example.org
```

Batch processing only works with non-interactive authentication.

21 http://man.openbsd.org/ssh_config.5
22 http://man.openbsd.org/sshd_config.5
23 http://man.openbsd.org/sftp.1
24 http://man.openbsd.org/ssh.1
25 http://man.openbsd.org/sftp.1
26 http://man.openbsd.org/sftp.1
27 http://man.openbsd.org/ssh.1
28 http://man.openbsd.org/sftp.1
29 http://man.openbsd.org/sftp.1
30 http://man.openbsd.org/ssh_config.5

6.3 The SCP client

scp(1)[31] is used for encrypted transfers of files between hosts and is used a lot like regular cp(1)[32]. Since 9.0[28] it is based on the SFTP protocol under the hood, while the old version prior to that was inspired by the Berkeley Software Distribution (BSD) Remote Copy Protocol (RCP) for which it has been a secure replacement. Both the old and the new versions use SSH to encrypt the connection.

The scp(1)[33] client, unlike the SFTP client, is not based on any formal standard. It has aimed at doing more or less what old **rcp** did and responded the same way. Since with it the same program must be used at both ends of the connection and interoperability is required with other implementations of SSH. Changes in functionality would probably break that interoperability, so new features are more likely to be added to sftp(1)[34] if at all. Thus, it is best to lean towards using sftp(1)[35] instead when possible. Again, recent versions of scp(1)[36] are a front end the SFTP protocol anyway.

Copy from remote to local:

```
$ scp myaccount@sftp.example.org:*.txt .
```

Copy from local to remote, recursively:

```
$ scp -r /etc/ myaccount@sftp.example.org:.
```

Being a front end for the SFTP protocol now, the new scp(1)[37] client can cover most but not all of the behavior of the old client and is not quite bug-for-bug compatible. One noticeable change which is fixed on the server side in OpenSSH 8.7 or later was the absence of tilde (~) expansion. So the "expand-path@openssh.com" protocol extension has been added to support this. Another area where there can potentially be trouble is when there are shell meta-characters, such as * or ?, in the file names.

See also the section for the SFTP client above.

6.4 GUI Clients

There are a great many graphical utilities that support SFTP and SSH. Many started out as transfer utilities with the outdated legacy protocol FTP and grew with the times to include SSH and SFTP support. Sadly, many retain the epithet FTP program despite modernization. Others are more general file managers that include SFTP support as one means of network transparency. Most if not all provide full SFTP support including Kerberos authentication.

Below is a partial list to give an idea of the range of options available.

31 http://man.openbsd.org/scp.1
32 http://man.openbsd.org/cp.1
33 http://man.openbsd.org/scp.1
34 http://man.openbsd.org/sftp.1
35 http://man.openbsd.org/sftp.1
36 http://man.openbsd.org/scp.1
37 http://man.openbsd.org/scp.1

Bluefish is a website management tool and web page editor with built in support for SFTP. Closed source competitors XMetaL and Dreamweaver are said to have at least partial support for SFTP. No support for SFTP is available for Quanta+ or Kompozer as of this writing. [38]

Cyberduck is a remote file browser for the Macintosh. It supports an impressive range of protocols in addition to SFTP. [39]

Dolphin is a highly functional file manager for the KDE desktop, but can also be run in other desktop environments. It includes SFTP support

Fetch, by Fetch Softworks, is a reliable and well-known SFTP client for the Macintosh. It has been around since 1989 and started life as just an FTP client. It has many useful features combined with ease of use. It is closed source, but academic institutions are eligible for a free of charge site license. [40]

Filezilla is presented as a FTP utility, but it has built in support for SFTP. It is available for multiple platforms under the Free Software license, the GPL. [41]

FireFTP is a SFTP plugin for Mozilla Firefox. Though it is presented as an FTP add-on, it supports SFTP. It is available under both the MIT license and the GPL. [42]

Fugu, developed by the University of Michigan research systems unix group, is a graphical front-end for SFTP on the Macintosh. [43]

gFTP is a multi-threaded file transfer client [44]

JuiceSSH is an SSH Client for Android/Linux. It uses the jsch[45] Java implementation of SSH2. [46]

Konqueror is a file manager and universal document viewer for the KDE desktop, but can also be run in other environments. It includes SFTP support. [47]

lftp is a file transfer program that supports multiple protocols. [48]

Midnight Commander is a visual file manager based on a text interface and thus usable over a terminal or console. It includes SFTP support. [49]

Nautilus is the default file manager for the GNOME desktop, but can also be run in other environments. It includes SFTP support

38 http://bluefish.openoffice.nl/
39 http://cyberduck.ch/
40 http://fetchsoftworks.com/fetch/
41 https://filezilla-project.org/
42 http://fireftp.mozdev.org/
43 http://rsug.itd.umich.edu/software/fugu/
44 http://www.gftp.org/
45 http://www.jcraft.com/jsch/
46 https://juicessh.com/
47 http://www.konqueror.org/
48 http://lftp.yar.ru/
49 https://midnight-commander.org/

PCManFM is an extremely fast, lightweight, yet feature-rich file manager with tabbed browsing which is the default for LXDE. It includes SFTP support. [50]

PuTTY is another FOSS implementation of Telnet and SSH for both legacy and Unix platforms. It is released under the MIT license and includes an SFTP client, PSFTP, in addition to an xterm terminal emulator and other tools like a key agent, Paegent. It is written and maintained primarily by Simon Tatham. [51]

Remmina is a remote desktop client written in GTK+ which supports multiple network protocols, including SSH. [52]

RemoteShell is the default SSH client for MorphOS, written in C using the GUI library Magic User Interface (MUI). The operating system also contains the command-line tools ssh(1)[53], scp(1)[54] and sftp(1)[55]. [56]

SecPanel is a GUI for managing and running SSH and scp connections. It is not a new implementation of the protocol or software-suite, but sits on top of either of the SSH software-suites [57]

Thunar is the default file manager for the XFCE desktop. It includes SFTP support. [58]

Transfer is the default SFTP client for MorphOS, written in C using the GUI library Magic User Interface (MUI). [59]

Yafc is Yet Another FTP Client and despite the name supports SFTP. [60]

50 http://wiki.lxde.org/en/PCManFM
51 http://www.chiark.greenend.org.uk/~sgtatham/putty/download.html
52 http://www.remmina.org/
53 http://man.openbsd.org/ssh.1
54 http://man.openbsd.org/scp.1
55 http://man.openbsd.org/sftp.1
56 http://www.morphos-team.net
57 http://themediahost.de/secpanel/
58 http://docs.xfce.org/xfce/thunar/start
59 http://www.morphos-team.net
60 http://yafc.sourceforge.net/

7 Client Configuration Files

Client configuration files can be per user or system wide, with the former taking precedence over the latter and run-time arguments in the shell overriding both. In these configuration files, one parameter per line is allowed. The syntax is the parameter name followed by its value or values. Empty lines and lines starting with the hash (#) are ignored. An equal sign (=) can be used instead of whitespace between the parameter name and the values. Values are case-sensitive, but parameter names are not. The first value assigned is used. For key files, the format is different.

With either type of file, there is no substitute for reading the relevant manual pages on the actual systems involved, especially because they match the specific versions which are in use.

7.1 System-wide Client Configuration Files

System-wide client files set the default configuration for all users of OpenSSH clients on that system. These defaults can be overridden in most cases by the user's own default settings in a local configuration file. Both can be overridden, in many cases, by specifying various options or parameters at run time. The prioritization is as follows:

1. run time arguments via the shell
2. user's own configuration
3. system-wide configuration

The first value obtained is used. The user's own configuration file and the system-wide configuration file can also point to additional configuration files to be included using the **Include** directive starting with OpenSSH 7.3. The **Include** directive can be specified anywhere in the configuration file even inside a **Match** or **Host** block. Care should be used when nesting configurations.

7.1.1 */etc/ssh/ssh_config*

This file defines all the default settings for the client utilities for all users on that system. It must be readable by all users. The configuration options are described in detail in ssh_config(5)[1].

Below a shortcut is made for connecting to *arc.example.org*.

1 `http://man.openbsd.org/ssh_config.5`

```
Host arc
        Port 2022
        HostName arc.example.org
        User fred
        IdentityFile ~/.ssh/id_rsa_arc
```

So with that configuration, it is enough to enter `ssh arc` and the rest of the information gets filled in automatically.

7.1.2 */etc/ssh/ssh_known_hosts*

This contains the system-wide list of known host keys used to verify the identity of the remote host and thus hinder impersonation or eavesdropping. This file should be prepared by the system administrator to contain the public host keys of all necessary hosts. It should be world-readable.

See **~/.ssh/known_hosts** below for more explanation or see sshd(8)[2] for further details of the format of this file.

7.1.3 */etc/ssh/sshrc*

This file resides on the server and programs in this file are executed there by ssh(1)[3] when the user logs in, just before the user's shell or designated program is started. It is not run as root, but instead as the user who is logging in. See the sshd(8)[4] manual page in the section "SSHRC" for more information. If it sends anything to **stdout** that will interfere with SFTP sessions, among others. So if any output is produced at all, it should be sent to **stderr** or else a log file.

7.2 User-specific Client Configuration Files

Users can override the default system-wide client settings and choose their own defaults. For situations where the same change is made repeatedly it is recommended to add it to the user's local configuration.

7.2.1 Client-Side Files

These files reside on the client machine.

~/.ssh/config

The user's own configuration file which, where applicable, overrides the settings in the global client configuration file, **/etc/ssh/ssh_config**. The configuration options are described in detail in ssh_config(5)[5].

2 http://man.openbsd.org/sshd.8
3 http://man.openbsd.org/ssh.1
4 http://man.openbsd.org/sshd.8
5 http://man.openbsd.org/ssh_config.5

This file must *not* be accessible to other users in any way. Set strict permissions: read/write for the user, and not accessible by others. It may group-writable if and only if that user is the only member of the group in question.

Local Override of Client Defaults
The file is usually named **~/.ssh/config**. However, a different configuration file can be specified at runtime using the **-F** option. General options intended to apply to all hosts can be set by matching all hosts and should be done at the end of the configuration file. The first match takes precedence, therefore more specific definitions must come first and more general overrides at the end of the file.

```
Host server1
        ServerAliveInterval    200
        HostName    203.0.113.76

Host *
        ExitOnForwardFailure    yes
        Protocol    2
        ServerAliveInterval    400
```

Options given as runtime arguments will override even those in the configuration file. However, not all options can be set or overriden by the user. Those options which may not be set or overridden will be ignored.

~/.ssh/known_hosts

This file is local to the user account and contains the known keys for remote hosts. Often these are collected from the hosts when connecting for the first time, but they can be added manually. As with those keys stored in the global file, **/etc/ssh/ssh_known_hosts** , these keys are used to verify the identity of the remote host, thus protecting against impersonation or man-in-the-middle attacks. With each subsequent connection the key will be compared to the key provided by the remote server. If there is a match, the connection will proceed. If the match fails, ssh(1)[6] will fail with an error message. If there is no key at all listed for that remote host, then the key's fingerprint will be displayed and there will be the option to automatically add the key to the file. This file can be created and edited manually, but if it does not exist it will be created automatically by ssh(1)[7] when it first connects to a remote host.

The **~/.ssh/known_hosts** file can use either hashed or clear text host names. Even with hashed names, it can still be searched using ssh-keygen(1)[8] using the **-F** option.

```
$ ssh-keygen -F server3.example.com
```

The default file to be searched will be **~/.ssh/known_hosts** and the key is printed if found. A different file can be searched using the **-f** option. If a key must be removed from the file, the **-R** option works similarly to search by host and then remove it if found even if the host name is hashed.

6 http://man.openbsd.org/ssh.1
7 http://man.openbsd.org/ssh.1
8 http://man.openbsd.org/ssh-keygen.1

```
$ ssh-keygen -R server4.example.com -f ~/.ssh/known_hosts
```

When a key is removed, it will then be appended to the file **~/.ssh/known_hosts.old** in case it is needed later. Again, see the manual page for sshd(8)[9] for the format of these **known_host** files.

If a non-default file is used with either **-F** or **-R** then the name including the path must be specified using **-f**. But **-f** is optional if the default file is intended.

If the global file **/etc/ssh/ssh_known_hosts** is used then it should be prepared by the system administrator to contain the public host keys of all necessary hosts and it should be world-readable.

Manually Adding Public Keys to ~/.ssh/known_hosts

Manually adding public host keys to **known_hosts** is a matter of adding one unbroken line per key. How the key is obtained is not important, as long as it is complete, valid, and **guaranteed to be the real key and not a fake**. The utility ssh-keyscan(1)[10] can fetch a key and ssh-keygen(1)[11] can be used to show the fingerprint for verification. See examples in the cookbook chapter on Public Key Authentication[12] for methods of verification. Again, the corresponding system-wide file is **/etc/ssh/ssh_known_hosts**

About the Contents of the known_hosts Files

The **known_hosts** file is for verifying the identity of other systems. ssh(1)[13] can automatically add keys to the user's file, but they can be added manually as well. The file contains a list of public keys for all the hosts which the user has connected to. It can also include public keys for hosts that the user plans to log into but are not already in the system-wide list of known host keys. Usually when connecting to a host for the first time, ssh(1)[14] adds the remote host's public key to the user's **known_hosts** file, but this behavior can be tuned.

The format is one public key or certificate per unbroken line. Each line in contains a host name, number of bits, exponent, and modulus. At the beginning of the line is either the host name or a hash representing the host name. An optional comment can follow at the end of the line. These can be preceded by an optional marker to indicate a certificate authority, if an SSH certificate is used instead of a SSH key. These fields are separated by spaces. It is possible to use a comma-separated list of hosts in the host name field if a host has multiple names or if the same key is used on multiple machines in a server pool. Here are two examples for hosts with the basic host names:

```
anoncvs.fr.openbsd.org,93.184.34.123 ssh-rsa AAAA...njvPw==
anoncvs.eu.openbsd.org ssh-rsa AAAAB3Nz...cTqGvaDhgtAhw==
```

9 http://man.openbsd.org/sshd.8
10 http://man.openbsd.org/ssh-keyscan.1
11 http://man.openbsd.org/ssh-keygen.1
12 https://en.wikibooks.org/wiki/OpenSSH/Cookbook/Public_Key_Authentication
13 http://man.openbsd.org/ssh.1
14 http://man.openbsd.org/ssh.1

Non-standard ports can be indicated by enclosing the host name with square brackets and following with a colon and the port number. Here are three examples referring to hosts listening for SSH on non-standard ports:

```
[ssh.example.org]:2222 ssh-rsa AAAAB3Nz...AKy2R2OE=
[127.0.0.2]:4922 ssh-rsa AAAAB4mV...1d6j=
[anga.funkfeuer.at]:2022,[78.41.115.130]:2022 ssh-rsa AAAAB...fgTHaojQ==
```

Host name patterns can be created using "*****" and "**?**" as wildcards and "**!**" to indicate negation.

Up to one optional marker per line is allowed. If present it must be either **@cert-authority** or **@revoked**. The former shows that the key is a certificate authority key, the latter flags the key as revoked and not acceptable for use.

See sshd(8)[15] for further details on the format of this file and ssh-keygen(1)[16] for managing the keys.

7.2.2 Server-Side Client Files

These client files reside on the server. By default they are kept in the user's directory. However, the server can be configured to look for them in other locations if needed.

~/.ssh/authorized_keys

authorized_keys is a one-key-per-line register of public ECDSA, RSA, and ED25519 keys that this account can use to log in with. The file's contents are not highly sensitive, but the recommended permissions are read/write for the user and not accessible by others. As always, the whole key including options and comments must be on a single, unbroken line.

```
ssh-rsa AAAAB3NzaC1yc2EAAA...41Ev521Ei2hvz7S2QNr1zAiVaOFy5Lwc8Lo+Jk=
```

Lines starting with a hash (#) are ignored and can be used as comments. Whitespace separates the key's fields, which are in sequence an optional list of login options, the key type (usually ssh-rsa or better like ecdsa-sha2-nistp256), the key itself encoded as base64, and an optional comment.

If a key is followed by an annotation, the comment does not need to be wrapped in quotes. It has no effect on what the key does or how it works. Here is an annotated key, the comment having been generated with the **-C** option ssh-keygen(1)[17]:

```
ssh-rsa AAAAB3NzaC1yc2EAAA...zAiVaOFy5Lwc8Lo+Jk=  Fred @ Project FOOBAR
```

Keys can be preceded by a comma-separated list of options to affect what happens upon successful login. The first key below forces the session to launch **tinyfugue** automatically, the second forcibly sets the PATH environment variable:

15 http://man.openbsd.org/sshd.8
16 http://man.openbsd.org/ssh-keygen.1
17 http://man.openbsd.org/ssh-keygen.1

```
command="/usr/bin/tinyfugue" ssh-rsa AAAAB3NzaC1yc2EAAA...OFy5Lwc8Lo+Jk=
environment="PATH=/bin:/usr/bin/:/opt/gtm/bin" ssh-rsa AAAAB3N...4Y2t1j=
```

The format of **authorized_keys** is described in the sshd(8)[18] manual page. Old keys should be deleted from the file when no longer needed. The server can specify multiple locations for **authorized_keys**. See the next section, Server-Side Client Key Login Options[19], for details.

~/.ssh/authorized_principals

By default this file does not exist. If it is specified in sshd_config(5)[20], it contains a list of names which can be used in place of the username when authorizing a certificate. This option is useful for role accounts, disjoint account namespaces and "user@realm"-style naming policies in certificates. Principals can also be specified in **authorized_keys**.

~/.ssh/environment

If the server is configured to accept user-supplied, automatic changes to environment variables as part of the login process, then these changes can be set in this file.

If the server, the environment file and an authorization key all try to change the same variable, the file **environment** takes precedence over what a key might contain. Either one will override any environment variables that might have been passed by ssh(1)[21] using **SendEnv**.

Authentication keys stored in **authorized_keys** can also be used to set variables. See also the **AcceptEnv** and **PermitUserEnvironment** directives in the manual page for sshd_config(5)[22].

~/.ssh/rc

This is a script which is executed by sh(1)[23] just before the user's shell or command is started. It is not run if **ForceCommand** is used. The script is run after reading the environment variables. The corresponding global file, **/etc/ssh/sshrc**, is not run if the user's **rc** script exists.

7.2.3 Local Account Public / Private Key Pairs

People might have a variety ECDSA, Ed25519, and RSA keys stored in the file system. Since version 8.2, two new key types ECDSA-SK and Ed25519-SK, along with corresponding certificate types are available for keys tied to FIDO/U2F tokens. Though individual accounts can maintain their own list of keys or certificates for authentication or to verify

18 http://man.openbsd.org/sshd.8
19 #Server-Side_Client_Key_Login_Options
20 http://man.openbsd.org/sshd_config.5
21 http://man.openbsd.org/ssh.1
22 http://man.openbsd.org/sshd_config.5
23 http://man.openbsd.org/sh.1

the identity of remote hosts in any directory, the most common location is in the **~/.ssh/** directory. The naming convention for keys is only a convention but recommended to follow anyway. Public keys usually have the same name as the private key, but with **.pub** appended to the name. Trouble can arise if the names of the public and private keys do not match. If there is more than one key pair, then ssh-keygen(1)[24] can use the **-f** option when generating keys to produce a useful name along with the **-C** option which embeds a relevant comment inside the key pair.

People, programs, and scripts can authenticate using a private key stored on the system, or even a private key fetched from a smartcard, if the corresponding public key is stored in **authorized_keys** on the remote system. The **authorized_keys** file is not highly sensitive, but the recommended permissions are read/write for the user, and not accessible by others. The private keys, however, are very sensitive and should not be readable or even visible to other accounts. They should never leave the client and should certainly never be put on the server. See the chapter on Public Key Authentication[25] for more discussion and examples.

The keys can be preceded by a comma-separated list of options. The whole key must be on a single, unbroken line. No spaces are permitted, except within double quotes. Any text after the key itself is considered a comment. The **authorized_keys** file is a one-key-per line register of public RSA, Ed25519, ECDSA, Ed25519-SK, and ECDSA-SK keys that can be used to log into a particular account. See the section above on the **authorized_keys** file for more discussion.

DSA is considered deprecated. The time has passed for DSA keys and they are no longer considered safe and should be replaced with better keys.

Per-account Host-based Authentication Configuration[26] is also possible using the **~/.shosts**, **~/.rhosts**, **~/.ssh/environment**, and **~/.ssh/rc** files.

Public Keys: ~/.ssh/id_ecdsa.pub ~/.ssh/id_ed25519.pub ~/.ssh/id_rsa.pub ~/.ssh/id_ecdsa-sk.pub ~/.ssh/id_ed25519-sk.pub ~/.ssh/id_ecdsa-sk_rk.pub ~/.ssh/id_ed25519-sk_rk.pub

These are only the default names for the public keys. Again, it can be a good idea to give more relevant names to keys. The ***-sk.pub** keys are those bound with a hardware security token and the ***-sk_rk.pub** keys are those generated from resident keys stored within hardware security token.

Public keys are mainly used on the remote server for key-based authentication. Public keys are not sensitive and are allowed to be readable by anyone, unlike the private keys, but don't need to be. A public key, minus comments and restriction options, can be regenerated from a private key if lost. So while it can be useful to keep backups of the public key, it is not essential unlike for private keys.

24 http://man.openbsd.org/ssh-keygen.1
25 https://en.wikibooks.org/wiki/OpenSSH/Cookbook/Public_Key_Authentication
26 https://en.wikibooks.org/wiki/OpenSSH/Cookbook/Host-based_Authentication

Private Keys: ~/.ssh/id_ecdsa ~/.ssh/id_ed25519 ~/.ssh/id_rsa ~/.ssh/id_ecdsa-sk ~/.ssh/id_ed25519-sk ~/.ssh/id_ecdsa-sk_rk ~/.ssh/id_ed25519-sk_rk

These are only the default names for private keys. Private keys are always considered sensitive data and should be readable only by the user and not accessible by others. In other words, they use mode 0600. The directory they are in should also have mode 0700 or 0500. If a private key file is accessible by others, ssh(1)[27] will ignore it.

It is possible to specify a passphrase when generating the key which will be used to encrypt the sensitive part of this file using AES128. Until version 5.3, the cipher 3DES was used to encrypt the passphrase. Old keys using 3DES that are given new passphrases will use AES128 when they are modified.

Private keys stored in hardware tokens as resident keys can be extracted and automatically used to generate their corresponding public key using ssh-keygen(1)[28] with the **-K** option. Such keys will default to being named **id_ecdsa-sk_rk** or **id_ed25519-sk_rk**, depending on the key type, though the file names can be changed after extraction. A passphrase can be assigned to the private key upon extraction from the token to a file.

While public keys can be generated from private keys, new private keys cannot be regenerated from public keys if the private keys are lost. Nor can a new passphrase be set if the current one is forgotten. Gone is gone, unlike the public keys, which can be regenerated from an existing private key if the private key is lost or forgotten then a whole new key pair must be generated and deployed.

7.2.4 Legacy Files

These files might be encountered on very old or out of date systems but not on up-to-date ones.

~/.shosts

~/.rhosts

.rhosts is a legacy from rsh containing a local list of trusted host-user pairs that are allowed to log in. Login requests matching an entry were granted access.

See also the global list of trusted host-user pairs, **/etc/hosts.equiv**

rhosts can be used as part of host-based authentication. Otherwise it is recommended not to use rhosts for authentication, there are a lot of ways to misconfigure the **.rhosts** file.

Legacy DSA Keys ~/.ssh/id_dsa ~/.ssh/id_dsa.pub

Deprecated DSA keys might be found named as **id_dsa** and **id_dsa.pub**, but regardless of the name any usage should be tracked down. Support for DSA both on the server and

27 http://man.openbsd.org/ssh.1
28 http://man.openbsd.org/ssh-keygen.1

client was discontinued in OpenSSH 7.0. If DSA keys are found, the pair should be removed and replaced with a better type of key.

Legacy SSH1 Protocol Keys ~/.ssh/identity ~/.ssh/identity.pub

The files **identity** and **identity.pub** were for SSH protocol version 1, and thus deprecated. If found they should be investigated as to what, if anything, uses them and why. Then once any remaining usage is resolved they should be removed and replaced with newer key types.

7.3 Mapping Client Options And Configuration Directives

Many run-time options for the SSH client have corresponding configuration directives and vice versa. The following is a quick overview. It is not a substitute for getting familiar with the relevant manual pages, ssh(1)[29] and ssh_config(5)[30] which are the relevant, authoritative, up-to-date resources on the matter.

Lookup Table of OpenSSH Client Options Versus Configuration Directives		
Directive	**Option**	**Description**
AddressFamily	**-4 / -6**	Limit connections to IPv4 or IPv6 only.
ForwardAgent	**-A / -a**	Forward or block forwarding from the authentication agent.
BindInterface	**-B**	Bind the outgoing connection to this network interface.
BindAddress	**-b**	Bind the outgoing connection to this network address.
Compression	**-C**	Specify whether to compress the data using gzip(1)[31].
Ciphers	**-c**	Specify which cipher to use.
DynamicForward	**-D**	Designate a local port to be forwarded, say for SOCKS5.
EscapeChar	**-e**	Specify an escape character for PTY sessions.
ForkAfterAuthentication	**-f**	Drop client to background right before command execution.
GatewayPorts	**-g**	Whether other hosts are allowed to connect to local forwarded ports.
PKCS11Provider	**-I**	Specify the path to the shared PKCS#11 library.
IdentityFile	**-i**	Specify a particular certificate or private key to use for authentication.
ProxyJump	**-J**	Connect to the destination via this host or hosts first.

29 https://man.openbsd.org/ssh.1
30 https://man.openbsd.org/ssh_config.5
31 http://man.openbsd.org/gzip.1

Lookup Table of OpenSSH Client Options Versus Configuration Directives		
GSSAPIAuthentication	**-K** / **-k**	Enable or disable Generic Security Services Application Program Interface (GSSAPI) authentication.
LocalForward	**-L**	Specify which local port or socket to forward to the specified remote system.
User	**-l**	Designate which account on the remote system to try.
ControlMaster	**-M**	Allow multiplexing of SSH sessions over a single TCP connection.
MACs	**-m**	Designate which message authentication code (MAC) algorithms to try.
SessionType	**-N** / **-s**	Invoke a designated subsystem or even prevent any command execution at all.
StdinNull	**-n**	Prevent reading from **stdin**.
Tag	**-P**	Tag for use within **Match**.
Port	**-p**	Connect to this port on the remote system.
LogLevel	**-q** / **-v**	Adjust the verbosity of logging messages from the client.
RemoteForward	**-R**	Specify which remote port or socket to forward to the specified local system.
ControlPath	**-S**	Designate the control socket for multiplexing over this connection.
RequestTTY	**-T** / **-t**	Prohibit or request a pseudo-TTY for the session.
ForwardX11	**-X** / **-x**	Enable or prohibit X11 forwarding.

As of version 8.7 the **-f**, **-N**, and **-n** options also have corresponding client configuration directives in ssh_config(5)[32].

7.4 Server-Side Client Key Login Options

The login options available for use in the local user authorized keys file might be overridden or blocked by the server's own settings. However, within that constraint, the following options can be used.

cert-authority

Specifies that the listed key is a certification authority (CA) trusted to validate signed certificates for user authentication. Certificates may encode access restrictions similar to key options. If both certificate restrictions and key restrictions are present, then the most restrictive union of the two is applied.

command="*program*"

32 https://man.openbsd.org/ssh_config.5

Specifies a program and its options to be executed when the key is used for authentication. This is a good way of forcing a program to restrict a key to a single, specific operation such as a remote backup. However, TCP and X11 forwarding are still allowed unless explicitly disabled elsewhere.

The program is run on a PTY if the client requests it, otherwise the default is to run without a TTY. The default, running without a TTY, provides an 8-bit clean channel. If the default has been changed, specify **no-pty** to get an 8-bit clean channel. If no programs are allowed, then use an empty string "" to prevent anything from running.

```
no-pty,command="" ssh-rsa AAAAB3NzaC1yc2EAAA...OFy5Lwc8Lo+Jk=
```

If only one program is allowed, with specific options, then it can be spelled out explicitly.

```
restrict,command="/usr/bin/svnserve -t --tunnel-user=fred" ssh-ed25519
 AAAAC3NzaC1lZDI1NT...skSUlrRPoLyUq
```

Quotes provided in the program's options must be escaped using a backslash. ('\')

```
command="sh -c \"mysqldump db1 -u fred1 -p\"" ssh-rsa
 AAAAB3NzaC1yc...Lwc8OFy5Lo+kU=
```

This option applies to execution of the shell, another program, or a subsystem. Thus any other programs specified by the user are ignored when **command** is present. However, the program originally specified by the client remains available as the environment variable **SSH_ORIGINAL_COMMAND**. That can be used by a script in a multiple-choice case statement, for example, to allow the account to select from a limited range of actions.

environment="*NAME=value*"

Sets the value of an environment variable when this key is used to log in. It overrides default values of the variable, if they exist. This option can be repeated to set multiple variables up to 1024 discrete names. First match wins in the case of repetition. This option is only allowed if the **PermitUserEnvironment** option is set in the SSH server's configuration. The default is that it is disabled. This option used to be disabled automatically when **UseLogin** was enabled, but **UseLogin** has been deprecated.

expiry-time="*timespec*"

Sets a date or date-time, either as a YYYYMMDD date or a YYYYMMDDHHMM[SS], after which the key will not be allowed to authenticate. Otherwise the key will be considered valid indefinitely. The system time zone is used.

from="*pattern-list*"

Either the canonical name of the remote host or its IP address required in addition to the key. Addresses and host names can be listed using a comma-separated list of patterns, see PATTERNS in ssh_config(5)[33] for more information on patterns, or use the CIDR address/masklen notation.

no-agent-forwarding / **agent-forwarding**

33 http://man.openbsd.org/ssh_config.5

This option forbids the authentication agent from forwarding the key when it is used for authentication. Alternately, it allows agent forwarding even if it was otherwise previously disabled by the **restrict** option.

no-port-forwarding / **port-forwarding**

Forbids TCP forwarding and any port forward requests by the client will return an error when this key is used for authentication. Alternately, override the **restrict** option and allow port forwarding. See also **permitopen**.

no-pty / **pty**

TTY allocation is prohibited and any request to allocate a PTY will fail. Alternately, TTY allocation is permitted, even if previously disabled by the **restrict** option.

no-touch-required

FIDO keys which have been created with the **-O no-touch-required** can use this method which makes the client skip the check for user presence.

no-user-rc / **user-rc**

Use the **no-user-rc** option in **authorized_keys** to disable execution of **~/.ssh/rc**. Alternately, use **user-rc** to override the **restrict** option.

no-X11-forwarding / **x11-forwarding**

Prevent X11 forwarding when this key is used for authentication and requests to forward X11 will return an error. Alternately, override the **restrict** option and allow X11 forwarding.

permitlisten="[*host*:]*port*"

permitopen="*host*:*port*"

The **permitlisten** setting limits remote port forwarding (`ssh -R`) to only the specified port and, optionally, host. In contrast, **permitopen** limits local port forwarding (`ssh -L`) to only the specified host and port. IPv6 addresses can be specified with an alternative syntax: host/port. Multiple **permitopen** or **permitlisten** options may be used and must be separated by commas. No pattern matching is performed on the specified host names, they must be literal host names or IP addresses. Can be used in conjunction with agent-forwarding.

principals="*name1*[,*name2*,...]"

Specify a list of names that may be used in place of the username when authorizing a certificate trusted via the **TrustedCAKeys** option described in sshd_config(5)[34].

restrict

Disable all options, such as TTY allocation, port forwarding, agent forwarding, user-rc, and X11 forwarding all at once. Specific options can then be explicitly allowed on an individual basis.

tunnel="*n*"

34 `http://man.openbsd.org/sshd_config.5`

Select a specific tun(4)[35] device on the server. Otherwise when a tunnel device is requested without this option the next available device will be used.

verify-required

Require user-verification, such as with a PIN, with FIDO keys.

7.5 Managing Keys

When working with keys there are some basic, hopefully common sense, actions that should take place to prevent problems. The two most beneficial approaches are to use sensible names for the key files and to embed comments. The **-f** option for ssh-keygen(1)[36] allows a custom name to be set. The **-C** option allows a comment to be embedded in both the public and private keys. With the comment inside the private key, it can be regenerated automatically if a replacement public key is ever made using the **-y** option.

Other than that, there are three main rules of thumb for managing keys:

- Keys should use strong passphrases. If autonomous logins are required, then the keys should be first loaded into an agent and used from there. See ssh-add(1)[37] to get started there. It uses ssh-agent(1)[38] which many systems have installed and some have running by default.
- Keys should always be stored in protected locations, even on the client side. This is especially important for private keys. The private keys should not have read permissions for any user or group other than their owner. They should also be kept in a directory that is not accessible by anyone other than the owner in order to limit exposure.
- Old and unused keys should be removed from the server. In particular, keys without a known, valid purpose should be removed and not allowed to accumulate. Using the comment field in the public key for annotation can help eliminate some of the confusion as to the purpose and owner once some time has passed. Along those lines, keys should be rotated at intervals. Rotation means generating new key pairs and removing the old ones. This gives a chance to remove old and unused keys. It is also an opportunity to review access needs, whether access is required and if so at what level.

Following the principle of least privilege can limit the chance for accidents or abuse. If a key is only needed to run a specific application or script, then its login options should be limited to just what is needed. See sshd(8)[39] for the "AUTHORIZED_KEYS FILE FORMAT" section on key login options. For root level access, it is important to remember to configure **/etc/sudoers** or **/etc/doas.conf** appropriately. Access there can be granted to a specific application and even limit that application to specific options.

35 http://man.openbsd.org/tun.4
36 http://man.openbsd.org/ssh-keygen.1
37 http://man.openbsd.org/ssh-add.1
38 http://man.openbsd.org/ssh-agent.1
39 http://man.openbsd.org/sshd.8

One major drawback to keys is that they never expire and are valid indefinitely in principle. In contrast, certificates can be assigned a validity interval with an end date, after which they can no longer be used.

8 The Server

The OpenSSH Server, sshd(8)[1], listens for connections from clients and starts a new process or two for each new incoming connection to handle key exchange, encryption, authentication, program execution, and data exchange. In the case of multiplexing, some processes are reused. It can run standalone and wait in the background, be run in the foreground, or it can be loaded on demand by any Internet services daemon.

Since version 8.2, the listening process title shown in ps(1)[2] also shows the number of connections pending authentication.

```
$ ps -p $(pgrep -u root sshd) -o pid,user,args
  PID USER     COMMAND
44476 root     sshd: /usr/sbin/sshd [listener] 0 of 10-100 startups (sshd)
```

Note that this is the number pending authentication, not the number which of those which have already been authenticated. Those each have their own separate handler process owned by the account which has authenticated.

8.1 sshd

sshd(8)[3] is the secure shell daemon and it listens for incoming connections. The standard port for ssh(1)[4] as specified by IANA is 22 [29]. If sshd(8)[5] does not listen to a privileged port, it does not have to be launched by root. However there are few, if any occasions where a non-standard port should be considered. sshd(8)[6] can be bound to multiple addresses or just certain ones. Multiple instances of sshd(8)[7], each with a different configuration, can be run on the same machine, something which may be useful on multi-homed machines. An absolute path must be given to launch sshd(8)[8], i.e. `/usr/sbin/sshd`

Configuration data is parsed first from the arguments and options passed by the shell, the user-specific file, and lastly the system-wide configuration file.

1 http://man.openbsd.org/sshd.8
2 http://man.openbsd.org/ps.1
3 http://man.openbsd.org/sshd.8
4 http://man.openbsd.org/ssh.1
5 http://man.openbsd.org/sshd.8
6 http://man.openbsd.org/sshd.8
7 http://man.openbsd.org/sshd.8
8 http://man.openbsd.org/sshd.8

sshd(8)[9] - The SSH daemon that permits you to log in.

sftp-server(8)[10] - SFTP server subsystem, started automatically by sshd(8)[11] when needed.

ssh-keysign(8)[12] - Helper program for hostbased authentication.

sshd_config(5)[13] - The server configuration file.

The sshd(8)[14] daemon can be made to parse the configuration file, test it for validity, and then report the effective configuration settings. This is done by running the extended test mode (**-T**). The extended test will print out the actual server settings. It can also report modifications to the settings through use of the **Match** directive when combined with the connection specification (**-C**) parameter. The options for **-C** are **user**, **host**, and **addr** . This is where **host** and **addr** refer to the host running sshd(8)[15] and the address from which the connection is being made, respectively.

The following will print out the configurations that will be applied if the user 'fred' tries to log in to the host *server.example.org* from the address *192.168.100.5*.

```
$ /usr/sbin/sshd -TC user=fred,host=server.example.org,addr=192.168.100.5
```

The output is long, so it might be sensible to pipe it through sort(1)[16] and a pager like less(1)[17]. See the section on Debugging a Server Configuration[18] for more options.

By default, login is allowed for all groups. However, if either **AllowGroups** or **AllowUsers** is specified, then all users or groups not listed are prohibited from logging in. The allow/deny directives are processed in the following order:

1. **DenyUsers**,
2. **AllowUsers**,
3. **DenyGroups**, and finally,
4. **AllowGroups**.

The first pattern matched takes effect, so if **AllowUsers** exists it will completely override **AllowGroups** regardless of the order in which they appear in the configuration file. So for the most flexibility, it is recommended to use **AllowGroups**. In contrast, **DenyUsers** and **DenyGroups** do not interfere with each other and may be used together. List group names or patterns of group names, separated by spaces. If specified, login is allowed or denied only for users who are members of a group that matches a group or pattern on the list. Only group or user names are valid; numerical group or user IDs are not recognized.

9 http://man.openbsd.org/sshd.8
10 http://man.openbsd.org/sftp-server.8
11 http://man.openbsd.org/sshd.8
12 http://man.openbsd.org/ssh-keysign.8
13 http://man.openbsd.org/sshd_config.5
14 http://man.openbsd.org/sshd.8
15 http://man.openbsd.org/sshd.8
16 http://man.openbsd.org/sort.1
17 http://man.openbsd.org/less.1
18 https://en.wikibooks.org/wiki/OpenSSH/Logging_and_Troubleshooting#Debugging_a_server_configuration

8.2 sshd under inetd / xinetd

An Internet services daemon is a server to launch other servers on demand. xinetd(8)[19] and inetd(8)[20] are two variants, either of which can be used to specify additional parameters and constraints, including running the launched service as a particular user and group. By having a single daemon active, which invokes others as needed, demands on the system can be reduced. Launching sshd(8)[21] this way means inetd(8)[22] waits for an incoming request, launches sshd(8)[23] and then when the SSH session is over, closes sshd(8)[24].

```
              Packet
 Internet --> Filter --> tcpwrappers --> (x)inetd --> sshd
             (firewall)  (aka tcpd)
```

Either can be used for additional logging such as successful or unsuccessful login, access restriction even including time of day, cpu priority, and number of connections. There are many more possibilities. See the manual pages for xinetd.conf(5)[25] or inetd.conf(5)[26] for a full overview of configuration options.

inetd(8)[27] was **tcpd**-aware and could make use of **tcpd'**s tcpwrappers to further control access or logging. So was sshd(8)[28] by itself, up through 6.6. See the manual pages for [ht ormation about how to use the configuration files **hosts.allow** and **hosts.deny**. Since 6.7, OpenSSH itself no longer supports tcpwrappers because current packet filters filters made it mostly redundant.

The two main disadvantages of using inetd(8)[29] or xinetd(8) are that there can be a slight increase in the delay during the start of the connection and that sshd(8)[30] must be configured to allow launching from the services daemon. The delay only affects the initial connection and thus does not get in the way of actual operation. An Internet services daemon should not be used for stateless services like HTTP and HTTPS, where every action is essentially a new connection. Again, see the manual page for xinetd.conf(5)[31] or inetd.conf(5)[32] for more details.

Example from xinetd.conf(5)[33]

```
service ssh
{
    socket_type     = stream
    protocol        = tcp
    wait            = no
```

19 http://linux.die.net/man/8/xinetd
20 http://man.openbsd.org/inetd.8
21 http://man.openbsd.org/sshd.8
22 http://man.openbsd.org/inetd.8
23 http://man.openbsd.org/sshd.8
24 http://man.openbsd.org/sshd.8
25 http://linux.die.net/man/5/xinetd.conf
26 http://man.openbsd.org/inetd.conf.5
27 http://man.openbsd.org/inetd.8
28 http://man.openbsd.org/sshd.8
29 http://man.openbsd.org/inetd.8
30 http://man.openbsd.org/sshd.8
31 http://linux.die.net/man/5/xinetd.conf
32 http://man.openbsd.org/inetd.conf.5
33 http://linux.die.net/man/5/xinetd.conf

```
    user            = root
    server          = /usr/sbin/sshd
    server_args     = -i
    per_source      = UNLIMITED
    log_on_failure  = USERID HOST
    # log_on_success  = PID HOST DURATION TRAFFIC EXIT
    # instances       = 10
    # nice            = 10
    # bind            = 192.168.0.100
    # only_from       = 192.168.0.0
    # access_times    = 08:00-15:25
    # no_access       = 192.168.54.0
    # no_access       += 192.168.33.0
    # banner          = /etc/banner.inetd.connection.txt
    # banner_success  = /etc/banner.inetd.welcome.txt
    # banner_fail     = /etc/banner.inetd.takeahike.txt
}
```

Example from inetd.conf(5)[34]

```
ssh    stream  tcp     nowait  root /usr/sbin/sshd -i
ssh    stream  tcp6    nowait  root /usr/sbin/sshd -i
```

There are several advantages with xinetd(8)[35] over inetd(8)[36] in capabilities but use-cases where either would be useful are rare.

8.3 The SFTP Server Subsystem

The SFTP subsystem first appeared in OpenBSD 2.8 / OpenSSH 2.3[30]. It is called by sshd(8)[37] as needed using the **Subsystem** configuration directive and not intended to operate standalone. There are two forms of the subsystem. One is the regular sftp-server(8)[38]. The other is an in-process SFTP server, which requires no support files when used with the **ChrootDirectory** directive. The **Subsystem** configuration directive can be used to pass options:

-d specifies an alternate starting directory for users, the default is the user's home directory. (First in 6.2)

```
Subsystem sftp internal-sftp -d /var/www
```

-e causes logging information to be sent to **stderr** instead of syslog(3)[39].

```
Subsystem sftp internal-sftp -e
```

34 http://man.openbsd.org/inetd.conf.5
35 http://linux.die.net/man/8/xinetd
36 http://man.openbsd.org/inetd.8
37 http://man.openbsd.org/sshd.8
38 http://man.openbsd.org/sftp-server.8
39 http://man.openbsd.org/syslog.3

-f specifies the syslog(3)[40] facility code that is used when logging messages from sftp-server(8)[41]. The possible values are: DAEMON, USER, AUTH, LOCAL0, LOCAL1, LOCAL2, LOCAL3, LOCAL4, LOCAL5, LOCAL6, LOCAL7.

```
Subsystem    sftp    /usr/libexec/sftp-server -f LOCAL0
```

-l Specifies which messages will be logged by sftp-server(8)[42]. The default is AUTH. The other possible values are: QUIET, FATAL, ERROR, INFO, VERBOSE, DEBUG, DEBUG1, DEBUG2, and DEBUG3. INFO and VERBOSE log transactions that sftp-server performs on behalf of the client. DEBUG and DEBUG1 are equivalent while DEBUG2 and DEBUG3 each specify higher levels of debugging output. Log levels DEBUG through DEBUG3 will violate user privacy and should not be used for regular operation. The default log level is ERROR. The actual path will vary depending on distro or operating system.

```
Subsystem    sftp    /usr/libexec/sftp-server -l VERBOSE
```

-p and **-P** specify whitelisted and blacklisted protocol requests, respectively. The comma separated lists are permitted or prohibited accordingly, the blacklist is applied first if both are used. **-Q** provides a list of protocol features supported by the server. All three are available as of version 6.5. The actual path will vary depending on distro or operating system.

In version 6.5 *requests* are the only protocol features queriable.

```
$ /usr/libexec/sftp-server -Q requests
```

-R places the SFTP subsystem in read-only mode. Attempts to change the filesystem, including opening files for writing, will fail.

-u overrides the user's default umask and explicitly sets the umask(2)[43] to be used for creating files and directories. See the manual page for syslog.conf(5)[44] for more information about log level or log facility. sshd(8)[45] must be able to access /**dev**/**log** for logging to work. Using the sftp-server(8)[46] subsystem in conjunction with the main SSH server's **ChrootDirectory** option therefore requires that syslogd(8)[47] establish a logging node inside the chrooted directory.

```
Subsystem sftp internal-sftp -u 0002
```

That sets the umask for the SFTP subsystem in OpenSSH 5.4 and later.

40 http://man.openbsd.org/syslog.3
41 http://man.openbsd.org/sftp-server.8
42 http://man.openbsd.org/sftp-server.8
43 http://man.openbsd.org/umask.2
44 http://man.openbsd.org/syslog.conf.5
45 http://man.openbsd.org/sshd.8
46 http://man.openbsd.org/sftp-server.8
47 http://man.openbsd.org/syslogd.8

8.4 Environment Variables

ssh(1)[48] and sshd(8)[49] set some environment variables automatically when logging in. Other variables can be explicitly defined by users in the **~/.ssh/environment** file if the file exists and if the user is allowed to change the environment. Variables can also be set on a key by key basis in the **authorized_keys** file, again only if the user is allowed to change the environment.

In **~/.ssh/environment**, the format **NAME=value** is used to set the variable. In **~/.ssh/authorized_keys** and **/etc/ssh/authorized_keys** the format is **environment="NAME=value"** For more information, see the **PermitUserEnvironment** and **AcceptEnv** configuration directives in sshd_config(5)[50] and the **SendEnv** directive in ssh_config(5)[51].

The following variables can be set by ssh(1)[52], depending on the situation.

DISPLAYIf X11 is tunneled, this is set so that the **DISPLAY** variable indicates the location of the X11 server. When it is automatically set by ssh(1)[53] it points to a value in the form *hostname*:*n*, where *hostname* indicates the host where the shell runs, and *n* is an integer greater than or equal to one. ssh(1)[54] uses this special value to forward X11 connections over the secure channel. The user should normally not set **DISPLAY** explicitly, as that will render the X11 connection insecure and will require the user to manually copy any required authorization cookies.

HOMEThe path of the user's home directory.

LOGNAMESynonym for USER. This is set for compatibility with systems that use this variable.

MAILThe path of the user's mailbox.

PATHThe default PATH, as specified when compiling ssh(1)[55].

SSH_ASKPASS If **DISPLAY** and **SSH_ASKPASS** are both set, and the SSH session does not have an associated terminal or pseudo-terminal, the program specified by **SSH_ASKPASS** will execute and open an X11 window to read the passphrase when one is needed. This is particularly useful when calling ssh(1)[56] from an xsession or related script. On some machines it may be necessary to redirect the input from **/dev/null** to make this work.

SSH_AUTH_SOCK The path on the client machine to tell ssh(1)[57] the UNIX-domain socket used to communicate with an SSH key agent.

48 http://man.openbsd.org/ssh.1
49 http://man.openbsd.org/sshd.8
50 http://man.openbsd.org/sshd_config.5
51 http://man.openbsd.org/sshd_config.5
52 http://man.openbsd.org/ssh.1
53 http://man.openbsd.org/ssh.1
54 http://man.openbsd.org/ssh.1
55 http://man.openbsd.org/ssh.1
56 http://man.openbsd.org/ssh.1
57 http://man.openbsd.org/ssh.1

SSH_CLIENT Identifies the client end of the connection. It contains three space-separated values: the client IP address, client port number and the server port number.

SSH_CONNECTION Identifies the client and server ends of the connection. The variable contains four space-separated values: client IP address, client port number, server IP address, and server port number.

SSH_ORIGINAL_COMMAND If the **ForceCommand** directive was used, or **Command="..."** in a key, then this variable contains the original command including the original options. It can be used to extract the original arguments.

SSH_TTY This is set to the name of the TTY (path to the device) associated with the current shell or command. If the current session has no TTY, this variable is not set.

SSH_USER_AUTH This will contain the name of a temporary file containing the authentication methods used for this particular session if **ExposeAuthInfo** is set in sshd_config(5)[58].

TZThis variable is set to indicate the present time zone if it was set when the daemon was started. The SSH daemon passes this value on to new connections.

USERSet to the name of the user logging in.

8.5 Pattern Matching in OpenSSH Configuration

A pattern consists of zero or more non-whitespace characters. An asterisk (*****) matches zero or more characters in a row, and a question mark (**?**) matches exactly one character. For example, to specify a set of declarations that apply to any host in the ".co.uk" set of domains in ssh_config(5)[59], the following pattern could be used:

```
Host *.co.uk
```

The following pattern would match any host in the 192.168.0.1 - 192.168.0.9 range:

```
Host 192.168.0.?
```

A pattern-list is a list of patterns separated by whitespace. The following list of patterns match hosts in both the ".co.uk" or ".ac.uk" domains.

```
Host *.co.uk *.ac.uk
```

Individual patterns by themselves or as part of a pattern-lists may be negated by preceding them with an exclamation mark (**!**). The following will match any host from *example.org* except for *gamma*.

```
Host *.example.org !gamma.example.org
```

Pattern lists in ssh_config(5)[60] do not use commas. Pattern lists in keys need commas.

58 http://man.openbsd.org/sshd_config.5
59 http://man.openbsd.org/ssh_config.5
60 http://man.openbsd.org/ssh_config.5

For example, to allow a key to be used from anywhere within an organisation except from the dialup pool, the following entry in **authorized_keys** could be used:

```
from="!*.dialup.example.com,*.example.com"
```

See also glob(7)[61]

61 http://man.openbsd.org/glob.7

9 Utilities

ssh-agent(1)[1] - An authentication agent that can store private keys.

ssh-add(1)[2] - A tool which adds or removes keys to or from the above agent.

ssh-keygen(1)[3] - A key generation tool.

ssh-keyscan(1)[4] - A utility for gathering public host keys from a number of hosts.

ssh-copy-id(1)[5] - Install a public key in a remote machine's **authorized_keys** register.

ssh-vulnkey(1) - Check a key against blacklist of compromised keys

9.1 ssh-agent

ssh-agent(1)[6] is a tool to hold private keys in memory for re-use during a session. Usually it is started at the beginning of a session and subsequent windows or programs run as clients to the agent. The environment variable **SSH_AUTH_SOCK** points applications to the socket used to communicate with the agent.

9.2 ssh-add

ssh-add(1)[7] is a tool to load key identities into an agent for re-use. It can also be used to remove identities from the agent. The agent holds the private keys used for authentication.

9.3 ssh-keyscan

ssh-keyscan(1)[8] has been part of the OpenSSH suite since OpenSSH version 2.5.1 and is used to retrieve public keys. Keys retrieved using ssh-keyscan(1)[9], or any other method, must be verified by checking the key fingerprint to ensure the authenticity of the key and reduce the possibility of a man-in-the-middle attack. The default is to request a ECDSA

1 http://man.openbsd.org/ssh-agent.1
2 http://man.openbsd.org/ssh-add.1
3 http://man.openbsd.org/ssh-keygen.1
4 http://man.openbsd.org/ssh-keyscan.1
5 http://linux.die.net/man/1/ssh-copy-id
6 http://man.openbsd.org/ssh-agent.1
7 http://man.openbsd.org/ssh-add.1
8 http://man.openbsd.org/ssh-keyscan.1
9 http://man.openbsd.org/ssh-keyscan.1

key using SSH protocol 2. David Mazieres wrote the initial version of ssh-keyscan(1)[10] and Wayne Davison added support for SSH protocol version 2.

9.4 ssh-keygen

ssh-keygen(1)[11] is for generating key pairs or certificates for use in authentication, update and manage keys, or to verify key fingerprints. It works with SSH keys and can do the following activities:

- generate new key pairs, either ECDSA, Ed25519, RSA, ECDSA-SK or Ed25519-SK.
- remove keys from known hosts
- regenerate a public key from a private key
- change the passphrase of a private key
- change the comment text of a private key
- show the fingerprint of a specific public key
- show ASCII art fingerprint of a specific public key
- load or read a key to or from a smartcard, if the reader is available

If the legacy protocol, SSH1, is used, then ssh-keygen(1)[12] can only generate RSA keys. However, SSH1 is long since deprecated and the systems should be re-tooled if found in use. Also, although DSA keys can be generated, they are deprecated and should be replaced if found.

One important use for key fingerprints is when connecting to a machine for the first time. A fingerprint is a hash or digest of the public key. Fingerprints can be transferred out of band and loaded into either the **~/.ssh/known_hosts** or **/etc/ssh/ssh_known_hosts** files in advance of the first connection. The verification data for the key should be sent out of band. It can be sent ahead of time by post, fax, SMS or a phone call instead or otherwise communicated in some way such that you can be sure it is authentic and unchanged.

```
$ ssh -l fred zaxxon.example.org
The authenticity of host 'zaxxon.example.org (203.0.113.114)' can't be
 established.
RSA key fingerprint is SHA256:DnCHntWa4jeadiUWLUPGg9FDTAopFPR0c5TgjU/iXfw.
Are you sure you want to continue connecting (yes/no)?
```

If you see that message and the key's fingerprint matches the one you were given in advance, then the connection is probably good. If you see that and the key's fingerprint is different than what you were given in advance, then stop and disconnect and get on the phone or VoIP to work out the mistake. Once the SSH client has accepted the key from the server, it is saved in **known_hosts**.

```
$ ssh -l fred galaga.example.org
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@    WARNING: REMOTE HOST IDENTIFICATION HAS CHANGED!     @
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
IT IS POSSIBLE THAT SOMEONE IS DOING SOMETHING NASTY!
```

10 http://man.openbsd.org/ssh-keyscan.1
11 http://man.openbsd.org/ssh-keygen.1
12 http://man.openbsd.org/ssh-keygen.1

```
Someone could be eavesdropping on you right now (man-in-the-middle attack)!
It is also possible that a host key has just been changed.
The fingerprint for the ECDSA key sent by the remote host is
SHA256:QIWi4La8svQSf5ZYow8wBHN4tF0jtRlkIaLCUQRlxRI.
Please contact your system administrator.
Add correct host key in /home/fred/.ssh/known_hosts to get rid of this message.
Offending ECDSA key in /home/fred/.ssh/known_hosts:1
ECDSA host key for galaga.example.org has changed and you have requested strict
 checking.
Host key verification failed.
```

If you start to connect to a known host and you get an error like the one above, then either the first connection was to an impostor or the current connection is to an impostor, or something very foolish was done to the machine. Regardless, disconnect and don't try to log in. Contact the system administrator out of band to find out what is going on. [31] It is possible that the server was reinstalled, either the whole operating system or just the OpenSSH server, without saving the old keys. That would result in new keys being generated and explain their presence. Either way, check with the system administrator before connecting to be sure.

Hashed host names and addresses can be looked up in **known_hosts** using **-F**. Or else **-R** can be used to delete them.

```
$ ssh-keygen -F sftp.example.org -f ~/.ssh/known_hosts
# Host sftp.example.org found: line 7 type RSA
|1|slYCk3msDPyGQ8l0lq82IbUTzBU=|KN7HPqVnJHOFX5LFmTXS6skjK4o= ssh-rsa AAAAB3NzaC1
yc2EAAAABIwAAAIEA3cqqA6fZtgexZ7+4wxoLN1+YDvPfBtt4/m+N/RI8o95CXqvqZMIQjuVarVKjwRwt
9pTJIVzf6bwjcNkrUx9dQqZNpNBk
cvBRdmd775opWCAfkHEueKxkNx3Kb1yitzOdUaFkRwfTsXAjh+NleBq2ofAfjowu/zzCnnbAKy2R2OE=
```

9.5 ssh-copy-id

ssh-copy-id is included in some distros to install a public key into a remote machine's **authorized_keys** file. It is a simple shell script and the **authorized_keys** file should still be checked manually after first login to verify that everything went ok and that the key was copied as it should be.

9.6 ssh-vulnkey

ssh-vulnkey was included in some versions of some GNU/Linux distros to check a key against a blacklist of compromised keys. The blacklist was made necessary when a broken version of OpenSSL was distributed by some distros[32], resulting in bad keys that were easily predicted and compromised. Keys made while that broken version was in use that are found to have been compromised cannot be repaired and must be replaced. The problem has since been fixed and new keys should be all right.

10 Third-party Utilities

autossh[1] - Automatically restart SSH sessions and tunnels

scanssh[2] - a scanner for SSH hosts and some kinds of proxies

sshfs[3] - a user-space file system client based on SFTP

sshfp[4] - generates SSHFP DNS records from **known_hosts** files or ssh-keyscan(1)

keychain[5] - re-use ssh-agent and/or gpg-agent between logins

rsync[6] - synchronizes files and directories using delta encoding

gstm[7] - a graphical front-end for managing SSH-tunneled port redirects

sslh[8] - a protocol demultiplexer

sshguard[9] - an intrusion detection system with packet filtering

ssh-audit[10] - identifies the server's banner, key exchange, encryption, MAC, compression, compatibility, and other information.

webcat[11] - can use websockets[33] for tunneling, is otherwise very similar to netcat[12] and curl[13].

10.1 scanssh

scanssh scans hosts and networks for running services[34]. It checks the version number of the server and displays the results in a list. It detects ssh, sftp and several kinds of SOCKS, HTTP, and telnet proxies.

Scan a small subnet for ssh servers:

1 https://www.harding.motd.ca/autossh/
2 http://manpages.debian.net/cgi-bin/man.cgi?query=scanssh
3 http://linux.die.net/man/1/sshfs
4 http://linux.die.net/man/1/sshfp
5 http://linux.die.net/man/1/keychain
6 http://linux.die.net/man/1/rsync
7 http://gstm.sourceforge.net/
8 https://www.rutschle.net/tech/sslh/README.html
9 http://www.sshguard.net/
10 https://github.com/arthepsy/ssh-audit
11 https://git.sr.ht/~rumpelsepp/webcat
12 http://man.openbsd.org/nc.1
13 https://linux.die.net/man/1/curl

```
$ sudo scanssh -n 22 -s ssh 192.168.100.32/26
```

Scan the same small network for SOCKS proxies:

```
$ sudo scanssh -s socks5,socks4 192.168.100.32/26
```

Variable scanning speeds can be set as well as random sampling. Open proxy detection scans to detect open proxies on common ports.

Scan 1000 hosts randomly selected from 172.16.1.1 through 172.31.254.254, at a rate of 200 per second :

```
$ sudo scanssh -r 200 -p random(1000)/172.16.0.0/12
```

The hosts and networks to be scanned can be either specified as an IPv4 address or an CIDR like IP prefix with ip address and network mask. Ports can be appended by adding a colon at the end of address specification. The sequence of hosts scanned is random, but that can be modified by the following two parameters, **random** and **split**:

random(n[,seed])/ selects a sample of *n* random addresses from the range specified as targets for scanning. *n* is the number of address to randomly create in the given network and *seed* is an optional seed for the pseudo random number generator. For example, it is possible to sample 10000 random IPv4 hosts from the Internet by specifying 'random(10000)/0.0.0.0/0' as the address.

split(s,e)/selects a specific segment of the address range for use. *e* specifies the number of segments in parallel and *s* is the segment number used by this particular scan. This can be used to scan from several hosts in parallel by scanning a different segment from each host.

-n Specifies the port numbers to scan. Ports are separated by commas. Each specified scanner is run for each port in this list. The default port is 22.

Scan for SSH servers on both port 22 and 2022:

```
$ sudo scanssh -s ssh -n 22,2022 192.168.0.0/24
```

10.2 sshfs

sshfs builds on the Filesystem in Userspace (FUSE) interface to allow non-privileged users to create a secure, reliable file system framework. It allows a remote file system to be mounted as a local folder by taking advantage of the SFTP subsystem. It uses SFTP to mount a directory from a remote server as a local directory. In that way, all use applications can interact with that directory and its contents as if it were local.As the name implies, this is done in user space and not the kernel as is usually required for file systems. FUSE has a stable API library and bindings to C, C++, and Java. In this case it is specifically the SFTP client that is run over ssh(1)[14] and is then mounted as a file system.

14 http://man.openbsd.org/ssh.1

See the Cookbook section on SFTP[15] for more regarding **sshfs**.

10.3 sshfp

sshfp generates SSHFP NS records using the public keys stored in a **known_hosts** file or provided by ssh-keyscan(1)[16] as a means to use DNS to publish SSH key fingerprints. That in turn allows DNSSEC lookups to verify SSH keys before use. SSHFP resource records in DNS are used to store fingerprint of SSH public host keys that are associated with the host names. A record itself consists of an algorithm number, fingerprint type, and the fingerprint of the public host key.

See RFC 4255[17] for details on SSHFP.

10.4 keychain

keychain is another manager for ssh-agent(1)[18] to allow multiple shells and processes, including cron(8)[19] jobs, to use the keys held by the agent. It is often integrated into desktop-specific tools like Apple Keychain on OS X or kdewallet for KDE. [20]

10.5 rsync

rsync is a file transfer utility to transfer files between computers very efficiently. It can run on top of SSH or use its own protocol. SSH is the default. [21]

See the Cookbook section on Automated Backup[22] for examples on using **rsync** live or in scripts.

10.6 gstm (Gnome SSH Tunnel Manager)

gstm is a graphical front-end for managing SSH connections and especially port forwarding. [23]

10.7 sslh

15 https://en.wikibooks.org/wiki/OpenSSH/Cookbook/SFTP
16 http://man.openbsd.org/ssh-keyscan.1
17 https://tools.ietf.org/html/rfc4255
18 http://man.openbsd.org/ssh-agent.1
19 http://man.openbsd.org/cron.8
20 http://www.funtoo.org/en/security/keychain/intro/
21 https://rsync.samba.org/
22 https://en.wikibooks.org/wiki/OpenSSH/Cookbook/Automated_Backup
23 http://sourceforge.net/projects/gstm/

sslh is a protocol demultiplexer. It accepts connections on specified ports and forwards them based on the first packet sent by the client. It can be used to share a single port between SSH, SSL, HTTP, OpenVPN, tinc, and XMPP. [24]

See also the section on Multiplexing[25] for a discussion with examples.

10.8 sshguard

sshguard is an intrusion prevention system. It monitors logs to detect undesirable patterns of activities and triggers corresponding packet filter rules for increasing periods of time. It can be used with a few other services besides SSH. [26]

10.9 ssh-audit

ssh-audit is a python script to gather information about SSH servers. It can identify banners used, key exchange, encryption, Message Authentication Code (MAC) algorithms, compression, compatibility settings, and several other security-related aspects. [27]

10.10 Additional Third Party Utilities

The following are useful in working with OpenSSH, but outside the scope of this book to go into detail. They are nevertheless worth mentioning enough to warrant a list:

netstat – Show network connections, routing tables, interface statistics, masquerade connections, and multicast memberships

nc or **netcat** – Netcat, the TCP/IP swiss army knife.

socat – SOcket CAT, a multipurpose relay similar to netcat.

nmap – Network exploration tool and security scanner.

tcpdump – Display network traffic real time.

telnet – Unencrypted interaction with another host.

pagsh – Creates a new credential cache sandbox and process authentication group (PAG).

nohup – Invoke a process that ignores HANGUP signals

sudo – Execute programs as another user

lftp – A handy interactive multi-protocol file transfer text-based client supporting SFTP.

curl – A multi-protocol file transfer text-based client supporting SCP and SFTP.

24 https://www.rutschle.net/tech/sslh/README.html
25 https://en.wikibooks.org/wiki/OpenSSH/Cookbook/Multiplexing#Multiplexing_HTTPS_and_SSH_Using_sslh
26 http://www.sshguard.net/
27 https://github.com/arthepsy/ssh-audit

tmux – A terminal multiplexer.

11 Logging and Troubleshooting

Both the OpenSSH client and server offer a lot of choice as to where the logs are written and how much information is collected.

A prerequisite for logging is having an accurate system clock using the Network Time Protocol, NTP, or equivalent, service which provides ongoing time synchronization with rest of the world. The more accurate the time stamp in the log is, the faster it is to coordinate forensics between machines or sites or service providers. If you have to contact outside parties like a service provider, progress can usually only be made with very exact times.

11.1 Server Logs

By default sshd(8)[1] sends logging information to the system logs using the log level INFO and the system log facility AUTH. So the place to look for log data from sshd(8)[2] is in /**var**/**log**/**auth.log**. These defaults can be overridden using the **SyslogFacility** and **LogLevel** directives. Below is a typical server startup entry in the authorization log.

```
Mar 19 14:45:40 eee sshd[21157]: Server listening on 0.0.0.0 port 22.
Mar 19 14:45:40 eee sshd[21157]: Server listening on :: port 22.
```

In most cases the default level of logging is sufficient, but during initial testing of new services or activities it is sometimes necessary to have more information. Debugging info usually goes to **stderr**. Starting with OpenSSH 7.6, **Match** blocks can set alternate log levels for specific conditions.

The log excerpt below show the same basic server start up with increased detail. Contrast the log level DEBUG1 below with the default above:

```
debug1: sshd version OpenSSH_6.8, LibreSSL 2.1
debug1: private host key #0: ssh-rsa
 SHA256:X9e6YzNXMmr1O09LVoQL1Cau2ej6TBUxi+Y590KVsds
debug1: private host key #1: ssh-dss
 SHA256:XcPAY4soIxU2IMtYmnErrVOjKEEvCc3l5hOctkbqeJ0
debug1: private host key #2: ecdsa-sha2-nistp256
 SHA256:QIWi4La8svQSf5ZYow8wBHN4tF0jtRlkIaLCUQRlxRI
debug1: private host key #3: ssh-ed25519
 SHA256:fRWrx5HwM7E5MRcMFTdH95KwaExLzAZqWlwULyIqkVM
debug1: rexec_argv[0]='/usr/sbin/sshd'
debug1: rexec_argv[1]='-d'
debug1: Bind to port 22 on 0.0.0.0.
Server listening on 0.0.0.0 port 22.
```

1 http://man.openbsd.org/sshd.8
2 http://man.openbsd.org/sshd.8

```
debug1: Bind to port 22 on ::.
Server listening on :: port 22.
```

And here is the same startup using the most verbose level, DEBUG3:

```
debug2: load_server_config: filename /etc/ssh/sshd_config
debug2: load_server_config: done config len = 217
debug2: parse_server_config: config /etc/ssh/sshd_config len 217
debug3: /etc/ssh/sshd_config:52 setting AuthorizedKeysFile .ssh/authorized_keys
debug3: /etc/ssh/sshd_config:86 setting UsePrivilegeSeparation sandbox
debug3: /etc/ssh/sshd_config:104 setting Subsystem sftp internal-sftp
debug1: sshd version OpenSSH_6.8, LibreSSL 2.1
debug1: private host key #0: ssh-rsa
 SHA256:X9e6YzNXMmr1OO9LVoQL1Cau2ej6TBUxi+Y590KVsds
debug1: private host key #1: ssh-dss
 SHA256:XcPAY4soIxU2IMtYmnErrVOjKEEvCc3l5hOctkbqeJ0
debug1: private host key #2: ecdsa-sha2-nistp256
 SHA256:QIWi4La8svQSf5ZYow8wBHN4tF0jtRlkIaLCUQRlxRI
debug1: private host key #3: ssh-ed25519
 SHA256:fRWrx5HwM7E5MRcMFTdH95KwaExLzAZqWlwULyIqkVM
debug1: rexec_argv[0]='/usr/sbin/sshd'
debug1: rexec_argv[1]='-ddd'
debug2: fd 3 setting O_NONBLOCK
debug1: Bind to port 22 on 0.0.0.0.
Server listening on 0.0.0.0 port 22.
debug2: fd 4 setting O_NONBLOCK
debug1: Bind to port 22 on ::.
```

Every failed login attempt is recorded, once the value in directive **MaxAuthTries** is exceeded the connection is broken. Below is a log excerpt showing how the default log looks after some failed attempts:

```
...
Mar 19 11:11:06 server sshd[54798]: Failed password for root from 122.121.51.193
 port 59928 ssh2
Mar 19 11:11:06 server sshd[54798]: Failed password for root from 122.121.51.193
 port 59928 ssh2
Mar 19 11:11:07 server sshd[54798]: Failed password for root from 122.121.51.193
 port 59928 ssh2
Mar 19 11:11:08 server sshd[54798]: Failed password for root from 122.121.51.193
 port 59928 ssh2
Mar 19 11:11:09 server sshd[54798]: Failed password for root from 122.121.51.193
 port 59928 ssh2
Mar 19 11:11:10 server sshd[54798]: Failed password for root from 122.121.51.193
 port 59928 ssh2
Mar 19 11:11:10 server sshd[54798]: error: maximum authentication attempts
 exceeded for root from 122.121.51.193 port 59928 ssh2 [preauth]
Mar 19 11:11:10 server sshd[54798]: Disconnecting authenticating user root
 122.121.51.193 port 59928: Too many authentication failures [preauth]
...
```

It is not usually a good idea to allow root login, at least not root login with authentication via password. Blocking password authentication for root simplifies log analysis greatly, and in particular it eliminates the time consuming question of who is trying to get in and why. People that need full root level access can gain it through su(1)[3] for general activities. Or for specific tasks which need root level access, those can be given those privileges through custom-made entries for **sudo(8)** or doas(1)[4]. Note that in those cases,

3 https://man.openbsd.org/su.1
4 https://man.openbsd.org/doas.1

only specific services and programs should be allowed, not blanket access which is an all too common misconfiguration seen with **sudo(8)**. Alternatively, a single-purpose key made using **forced-commands-only** could be used since some argue that providing extra means of privilege escalation, such as su(1)[5], **sudo(8)**, or doas(1)[6], is more dangerous than carefully providing remote root access through a key or certificate tied to a specific function.

11.1.1 Successful logins

By default, the server does not store much information about user transactions. That is a good thing. It is also a good thing to recognize when the system is operating as it should. So here is an example of a successful SSH login:

```
Mar 14 19:50:59 server sshd[18884]: Accepted password for fred from 192.0.2.60
 port 6647 ssh2
```

And here is an example using a key for authentication. It shows the key fingerprint as a SHA256 hash in base64.

```
Mar 14 19:52:04 server sshd[5197]: Accepted publickey for fred from 192.0.2.60
 port 59915 ssh2: RSA SHA256:5xyQ+PG1Z3CIiShclJ2iNya5TOdKDgE/HrOXr21IdOo
```

And here is an example of successful authentication with a user certificate. The certificate's identification string is "foobar" and the serial number is "9624". In this example the certificate is using ECDSA and the key itself is using Ed25519. The certificate, being a different key of its own, has a different SHA256 fingerprint from the authentication key itself.

```
May 15 16:28:17 server sshd[50140]: Accepted publickey for fred from 192.0.2.60
 port 44456 ssh2: ECDSA-CERT SHA256:qGl9KiyXrG6mIOo1CTO1oHUvod7Ngs5VMHM14DTbxzI
 ID foobar (serial 9624) CA ED25519
 SHA256:fZ6L7TlBLqf1pGWzkcQMQMFZ+aGgrtYgRM90XO0gzZ8
```

Prior to 6.8, the key's fingerprint was a hexadecimal MD5 hash.

```
Jan 28 11:51:43 server sshd[5104]: Accepted publickey for fred from 192.0.2.60
 port 60594 ssh2: RSA e8:31:68:c7:01:2d:25:20:36:8f:50:5d:f9:ee:70:4c
```

In older versions of OpenSSH prior to 6.3 the key fingerprint is completely missing from authentication logging.

```
Jan 28 11:52:05 server sshd[1003]: Accepted publickey for fred from 192.0.2.60
 port 20042 ssh2
```

Here is an example of password authentication for an SFTP session, using the server's internal-sftp subsystem. The logging for that subsystem set to INFO.

```
Mar 14 20:14:18 server sshd[19850]: Accepted password for fred from 192.0.2.60
 port 59946 ssh2
Mar 14 20:14:18 server internal-sftp[11581]: session opened for local user fred
 from [192.0.2.60]
```

5 https://man.openbsd.org/su.1
6 https://man.openbsd.org/doas.1

Here is an example of a successful SFTP login using an RSA key for authentication.

```
Mar 14 20:20:53 server sshd[10091]: Accepted publickey for fred from 192.0.2.60
 port 59941 ssh2: RSA SHA256:LI/TSnwoLryuYisAnNEIedVBXwl/XsrXjli9Qw9SmwI
Mar 14 20:20:53 server internal-sftp[31070]: session opened for local user fred
 from [192.0.2.60]
```

Additional data, such as connection duration, can be logged with the help of **xinetd**.

11.1.2 Logging Problems from SSH Certificate Authentication

Usually, not much information is given about which certificate failed, just why it failed authentication. Finding the account or actual certificate in question can require some sleuthing. Generally no client side information is disclosed and all investigation must occur server side.

If the authentication is attempted again by other means, such as a password, then when the connection is closed there will be a log entry noting which account was involved. That is because so early in the connection sequence the process ID for the disconnection is the same and the account name and the originating address is included, giving a bit of a clue in pursuit of a solution. Sometimes even the account name will be included.

```
May  5 16:31:38 server sshd[252]: Connection closed by authenticating user fred
 192.0.2.60 port 44470 [preauth]
```

However, if the connection is allowed to timeout without first making any other authentication attempts by some other means, then there will be nothing to go on except maybe the time of day.

```
May  5 16:33:00 server sshd[90593]: fatal: Timeout before authentication for
 192.0.2.60 port 44718
```

Below are some common examples of log entries for failed certificate-based log in attempts. There can be more than one problem with a certificate but only one error will get logged at a time.

Expired or Not-yet-valid Certificate

Certificates which have not yet become valid or which have already expired get a log entry as to the reason but neither the account or certificate involved.

```
May  5 16:35:20 server sshd[252]: error: Certificate invalid: expired
```

Above is an expired certificate, below is a certificate which has not yet become valid.

```
May  5 16:58:00 server sshd[90593]: error: Certificate invalid: not yet valid
```

Neither type of event gives more information.

Valid Certificate but Invalid Principal

Like with expired certificates, very little information is given about the actual account or certificate. Here the certificate was tried with the wrong account, one not listed among the certificate's principals.

```
May  5 17:29:52 server sshd[98884]: error: Certificate invalid: name is not a
 listed principal
May  5 17:29:56 server sshd[98884]: Connection closed by authenticating user
 fred 192.0.2.60 port 45114 [preauth]
```

If the client closes the connection on purpose, there may be some information in the connection closed entry.

Valid Certificate but Invalid Source Address

If the certificate is limited to connecting from specific addresses or host names, the log will complain if the connection comes from a different address or host and identify the incorrect source address.

```
May  5 17:48:54 server sshd[2420]: cert: Authentication tried for fred with
 valid certificate but not from a permitted source address (192.0.2.61).
May  5 17:48:54 server sshd[2420]: error: Refused by certificate options
```

However, it will not be possible to identify the specific certificate directly.

11.1.3 Logging SFTP File Transfers

SFTP file transfers can be logged using **LogLevel** INFO or VERBOSE. The log level for the SFTP server can be set in sshd_config(5)[7] separately from the general SSH server settings.

```
Subsystem internal-sftp -l INFO
```

By default the SFTP messages will also end up in *auth.log* but it is possible to filter these messages to their own file by reconfiguring the system logger, usually **rsyslogd(8)** or **syslogd(8)**. Sometimes this is done by changing the log facility code from the default of AUTH. Available options are LOCAL0 through LOCAL7, plus, less usefully, DAEMON and USER.

```
Subsystem internal-sftp -l INFO -f LOCAL6
```

If new system log files are assigned, it is important to remember them in log rotation, too. Again, the **Match** directive can be use to change the log level for certain connections.

The following log excerpts are generated from using the log level INFO. A session starts with an open and end with a close. The number in the brackets is the process id for the SFTP session and is the only way to follow a session through the logs.

7 http://man.openbsd.org/sshd_config.5

```
Oct 22 11:59:45 server internal-sftp[4929]: session opened for local user fred
 from [192.0.2.33]
...
Oct 22 12:09:10 server internal-sftp[4929]: session closed for local user fred
 from [192.0.2.33]
```

Here is an SFTP upload of a small file of 928 bytes named **foo** to the home directory for user 'fred'.

```
Oct 22 11:59:50 server internal-sftp[4929]: open "/home/fred/foo" flags
 WRITE,CREATE,TRUNCATE mode 0664
Oct 22 11:59:50 server internal-sftp[4929]: close "/home/fred/foo" bytes read 0
 written 928
```

And a directory listing in the same session in the directory /**var**/**www**.

```
Oct 22 12:07:59 server internal-sftp[4929]: opendir "/var/www"
Oct 22 12:07:59 server internal-sftp[4929]: closedir "/var/www"
```

And lastly here is a download of the same small 928-byte file called **foo** from the home directory for the user 'fred'.

```
Oct 22 12:08:03 server internal-sftp[4929]: open "/home/fred/foo" flags READ
 mode 0666
Oct 22 12:08:03 server internal-sftp[4929]: close "/home/fred/foo" bytes read
 928 written 0
```

Successful transfers will be noted by a *close* message. Attempts to download (open) files that do not exist will be followed by a *sent status No such file* message on a line of its own instead of a *close*. Files that exist but that the user is not allowed to read will create a *sent status Permission denied* message.

11.1.4 Logging Chrooted SFTP

Logging with the built-in **sftp-subsystem** inside a chroot jail, defined by **ChrootDirectory**, needs a **./dev/log** node to exist inside the jail. This can be done by having the system logger such as syslogd(8)[8] add additional log sockets inside the chrooted directory when starting up. On some systems that is as simple as adding more flags, like "**-u -a /chroot/dev/log**", in /**etc**/**rc.conf.local** or whatever the equivalent startup script may be.

Here is an example of an SFTP login with password to a chroot jail using log level DEBUG3 for the SFTP-subsystem. The log shows a file upload:

```
Jan 28 12:42:41 server sshd[26299]: Connection from 192.0.2.60 port 47366
Jan 28 12:42:42 server sshd[26299]: Failed none for fred from 192.0.2.60 port
 47366 ssh2
Jan 28 12:42:44 server sshd[26299]: Accepted password for fred from 192.0.2.60
 port 47366 ssh2
Jan 28 12:42:44 server sshd[26299]: User child is on pid 21613
Jan 28 12:42:44 server sshd[21613]: Changed root directory to "/home/fred"
Jan 28 12:42:44 server sshd[21613]: subsystem request for sftp
```

8 http://man.openbsd.org/syslogd.8

```
Jan 28 12:42:44 server internal-sftp[2084]: session opened for local user fred
 from [192.0.2.60]
Jan 28 12:42:58 server internal-sftp[2084]: open "/docs/somefile.txt" flags
 WRITE,CREATE,TRUNCATE mode 0644
Jan 28 12:42:58 server internal-sftp[2084]: close ‘‘/docs/somefile.txt’’ bytes
 read 0 written 400
```

Remember that SFTP is a separate subsystem and that like the file creation mode, the log level and log facility are set separately from the SSH server in sshd_config(5)[9]:

```
Subsystem internal-sftp -l ERROR
```

11.1.5 Logging Stability of Client Connectivity

When **ClientAliveInterval** is set in the server's configuration, the server makes periodic probes of the clients which have established connections. At normal log levels, these are not noted in the log until something goes wrong.

If the **ClientAliveInterval** is exceeded more times in a row than allowed by **ClientAliveCountMax** the client is officially declared disconnected and the connection dropped. At the default log level of INFO a brief message is logged, identifying the client which has been dropped.

```
Sep  6 14:42:08 eee sshd[83709]: packet_write_poll: Connection from 192.0.2.97
 port 57608: Host is down
```

At log level DEBUG, the client's responses to the polls will be logged by the server showing that the session is still connected.

```
Sep  6 14:27:52 eee sshd[9075]: debug1: Got 100/147 for keepalive
```

Log level DEBUG2 and DEBUG3 will give even more information about the connection. However, even at log level DEBUG3, the specific client being polled will not be identified directly in the log messages and will have to be inferred from the process id of the daemon if such information is needed.

```
Sep  6 14:30:59 eee sshd[73960]: debug2: channel 0: request
 keepalive@openssh.com confirm 1
Sep  6 14:30:59 eee sshd[73960]: debug3: send packet: type 98
Sep  6 14:30:59 eee sshd[73960]: debug3: receive packet: type 100
Sep  6 14:30:59 eee sshd[73960]: debug1: Got 100/22 for keepalive
```

Again, when the **ClientAliveCountMax** is exceeded, the connection is broken after the final failure of the client to respond. Here is how that looks with the log level set to DEBUG2.

```
Sep  6 14:17:55 eee sshd[15780]: debug2: channel 0: request
 keepalive@openssh.com confirm 1
Sep  6 14:17:55 eee sshd[15780]: debug1: Got 100/22 for keepalive
Sep  6 14:18:37 eee sshd[15780]: debug2: channel 0: request
```

9 http://man.openbsd.org/sshd_config.5

```
 keepalive@openssh.com confirm 1
Sep  6 14:18:37 eee sshd[15780]: packet_write_poll: Connection from 192.0.2.97
 port 57552: Host is down
Sep  6 14:18:37 eee sshd[15780]: debug1: do_cleanup
Sep  6 14:18:37 eee sshd[48675]: debug1: do_cleanup
Sep  6 14:18:37 eee sshd[48675]: debug1: session_pty_cleanup: session 0 release
 /dev/ttyp0
```

The directives **ClientAliveInterval** and **ClientAliveCountMax** normally apply to all clients connecting to the server. However, they can be used inside a **Match** block and thus applied only to specific connections.

11.1.6 Logging Revoked Keys

If the **RevokedKeys** directive is used to point to a list of public keys that have been revoked, sshd(8)[10] will make a log entry when access is attempted using a revoked key. The entry will be the same whether a plaintext list of public keys is used or if a binary Key Revocation List (KRL) has been generated.

If password authentication is allowed, and the user tries it, then after the key authentication fails there will be a record of password authentication.

```
Mar 14 20:36:40 server sshd[29235]: error: Authentication key RSA
 SHA256:jXEPmu4thnubqPUDcKDs31MOVLQJH6FfF1XSGT748jQ revoked by file
 /etc/ssh/ssh_revoked_keys
...
Mar 14 20:36:45 server sshd[29235]: Accepted password for fred from 192.0.2.10
 port 59967 ssh2
```

If password authentication is not allowed, sshd(8)[11] will close the connection as soon as the key fails.

```
Mar 14 20:38:27 server sshd[29163]: error: Authentication key RSA
 SHA256:jXEPmu4thnubqPUDcKDs31MOVLQJH6FfF1XSGT748jQ revoked by file
 /etc/ssh/ssh_revoked_keys
...
```

The account trying the revoked key remains a mystery though, so it will be necessary to try to look up the key by its fingerprint from your archive of old keys using `ssh-keygen -lf` and read the key's comments. Although if a valid account cancels the connection without trying a password after the key attempt fails, the usual message will still be posted to the log.

```
Mar 14 20:44:04 server sshd[14352]: Connection closed by authenticating user
 fred 192.0.2.237 port 55051 [preauth]
...
```

That may provide some clue and allow filtering with a short AWK script, if the messages are all in the same log file.

10 http://man.openbsd.org/sshd.8
11 http://man.openbsd.org/sshd.8

```
$ awk '/revoked by file/ {
        pid[$5]++; key[$5]=$9; hash[$5]=$10; next;
    }
    pid[$5] && /closed by authenticating user/ {
        print key[$5], hash[$5], $10, $11;
        delete key[$5]; delete hash[$5]; delete pid[$5];
    }' /var/log/authlog
```

Similarly, if the client makes no attempt at logging in and just times out, the message will say just that.

```
Mar 18 21:40:25 server sshd[9942]: fatal: Timeout before authentication for
 198.51.100.236 port 53728
...
```

On the client side, no warning or error will be given if a revoked key is tried. It will just fail and the next key or method will be tried.

11.1.7 Brute force and Hail Mary attacks

It's fairly common to see failed login attempts almost as soon as the server is connected to the net. Brute force attacks, where one machine hammers on a few accounts trying to find a valid password, are becoming rare. In part this is because packet filters, like NFables for Linux and PF for the BSDs, can limit the number and rate of connection attempts from a single host. The server configuration directive **MaxStartups** can limit the number of simultaneous, unauthenticated connections.

```
...
Mar 18 18:54:44 server sshd[54939]: Failed password for root from
 201.179.249.231 port 52404 ssh2
Mar 18 18:54:48 server sshd[54939]: Failed password for root from
 201.179.249.231 port 52404 ssh2
Mar 18 18:54:49 server sshd[54939]: Failed password for root from
 201.179.249.231 port 52404 ssh2
Mar 18 18:54:49 server sshd[54939]: error: maximum authentication attempts
 exceeded for root from 201.179.249.231 port 52404 ssh2 [preauth]
Mar 18 18:54:49 server sshd[54939]: Disconnecting authenticating user root
 201.179.249.231 port 52404: Too many authentication failures [preauth]
...
```

Note the "authenticating user" is present in the logs from OpenSSH 7.5 and onward when a valid user name is attempted. When an invalid user name is attempted, that is written too.

```
...
Mar 18 18:55:05 server sshd[38594]: Invalid user ubnt from 201.179.249.231 port
 52471
Mar 18 18:55:05 server sshd[38594]: Failed password for invalid user ubnt from
 201.179.249.231 port 52471 ssh2
Mar 18 18:55:09 server sshd[38594]: error: maximum authentication attempts
 exceeded for invalid user ubnt from 201.179.249.231 port 52471 ssh2 [preauth]
Mar 18 18:55:09 server sshd[38594]: Disconnecting invalid user ubnt
 201.179.249.231 port 52471: Too many authentication failures [preauth]
...
```

The way to deal with brute force attacks coming from a single machine or network is to customize the server host's packet filter to limit the attacks or even temporarily block

machines that overload the maximum number or rate of connections. Optionally, one should also contact the attacker's net block owner with the IP address and exact date and time of the attacks.

A kind of attack common at the time of this writing is one which is distributed over a large number of compromised machines, each playing only a small role in attacking the server.

To deal with Hail Mary attacks, contact the attacker's net block owner. A form letter with a cut-and-paste excerpt from the log is enough if it gives the exact times and addresses. Alternately, teams of network or system administrators can work to pool data to identify and blacklist the compromised hosts participating in the attack.

Failed None For Invalid User

The SSH protocol specifies a number of possible authentication methods[35]. The methods `password`, `keyboard-interactive`, and `publickey` are fairly common. A lesser known authentication method is `none`, which will only succeed if the server requires no further authentication such as if **PermitEmptyPassword** is set and the account does not actually have a password[36]. Some SSH clients including OpenSSH's start by asking for `none` authentication and then use the list of remaining possible authentication methods to decide what to do next if that doesn't work.

```
...
Aug 10 19:09:05 server sshd[93126]: Failed none for invalid user admin from
 125.64.94.136 port 27586 ssh2
...
```

So in other words, that is a brute force attack trying the `none` authentication method. It is an attack which will only get into accounts which have been explicitly set with an empty password and furthermore have also been set up specifically to allow access by having both the authentication method of `none` and the **PermitEmptyPasswords** configuration directive enabled on the server. Most brute force attacks try only `password` authentication, and some of those even check for the `password` method and then give up if it is not available. Other attackers may just hammer away pointlessly even if the method is not available.

11.1.8 Connections Seemingly From 127.0.0.1, ::1, or Other localhost Addresses

When the SSH server is accessed via a reverse tunnel to another machine, the incoming connections will appear to be from the *localhost* address, which is usually *127.0.0.1* or *::1*.

```
Mar 23 14:16:16 server sshd[9265]: Accepted password for fred from 127.0.0.1
 port 40426 ssh2
```

If the port at the other end of the reverse tunnel is publicly accessible, it will be probed and possibly attacked. Because of the reverse tunnel, the attacks will then also appear to be coming from the server's own loopback address:

```
Mar 23 14:20:17 server sshd[5613]: Invalid user cloud from ::1 port 57404
Mar 23 14:20:21 server sshd[5613]: Failed password for invalid user cloud
 from ::1 port 57404 ssh2
Mar 23 14:20:26 server sshd[5613]: Failed password for invalid user cloud
```

```
 from ::1 port 57404 ssh2
Mar 23 14:20:32 server sshd[5613]: Failed password for invalid user cloud
 from ::1 port 57404 ssh2
Mar 23 14:20:35 server sshd[5613]: Connection closed by invalid user cloud ::1
 port 57404 [preauth]
```

Therefore the usual countermeasures like SSHGuard or Fail2Ban or other similar intrusion detection systems cannot be used because the *localhost* address is used by the tunnel for all login attempts, regardless of their real origins.

A partial solution would be to bind the incoming connections to a different IP address. The loopback interface would need an additional permanent address, an alias, for that. That alias could then be assigned when establishing the reverse tunnel:

```
$ ssh -R 2022:127.2.2.1:22 fred@vps.example.org
```

Thus it would designate the source address for all logins coming in over that tunnel. So, in that way, the alias would then show up in the logs instead of the default loopback address when that reverse tunnel is used:

```
Mar 23 18:00:13 server sshd[8525]: Invalid user cloud from 127.2.2.1 port 17271
Mar 23 18:00:15 server sshd[8525]: Failed password for invalid user cloud from
 127.2.2.1 port 17271 ssh2
Mar 23 18:00:19 server sshd[8525]: Failed password for invalid user cloud from
 127.2.2.1 port 17271 ssh2
Mar 23 18:01:23 server sshd[8525]: Failed password for invalid user cloud from
 127.2.2.1 port 17271 ssh2
Mar 23 18:01:26 server sshd[8525]: Connection closed by invalid user cloud
 127.2.2.1 port 17271 [preauth]
```

If the ports do not need to be available to the open Internet, a full solution would be just to ensure that they are not accessible from the outside. This would be done by not using the **-g** option on the client when making the reverse tunnel or else by setting the **GatewayPorts** directive in sshd_config(5)[12] back to the default of **no**, or both. The system's built-in packet filter can also be used. Then, even with the forwarded ports closed off from the outside, the **ProxyJump** option can still be used to skip through the jump host and use the setup for SSH access. However, since it is sometimes necessary that these ports be accessible to the outside world, this approach is not always an option.

11.2 Client Logging

The OpenSSH client normally sends log information to **stderr**. The **-y** option can be used to send output to the system logs, managed by syslogd(8)[13] or something similar. Client log verbosity can be increased or decreased by changing the **LogLevel** directive and the log facility changed with the **SyslogFacility** directive in ssh_config(5)[14]. Both require use of the **-y** run time option and do nothing without it.

12 http://man.openbsd.org/sshd_config.5
13 http://man.openbsd.org/syslogd.8
14 http://man.openbsd.org/ssh_config.5

Alternatively, instead of using the **-y** option, using the **-E** option sends log output to a designated file instead of **stderr**. Working with the system logs or separate log files are something which can be useful when running ssh(1)[15] in automated scripts. Below is an example of a connection to an interactive shell with the normal level of client logging:

```
$ ssh -l fred server.example.org
fred@server.example.org's password:
Last login: Thu Jan 27 13:21:57 2011 from 192.168.11.1
```

The same connection at the first level of verbosity gives lots of debugging information, 42 lines more.

```
$ ssh -v -l fred server.example.org
OpenSSH_6.8, LibreSSL 2.1
debug1: Reading configuration data /etc/ssh/ssh_config
debug1: Connecting to server.example.org [198.51.100.20] port 22.
debug1: Connection established.
debug1: key_load_public: No such file or directory
debug1: identity file /home/fred/.ssh/id_rsa type -1
debug1: key_load_public: No such file or directory
debug1: identity file /home/fred/.ssh/id_rsa-cert type -1
debug1: key_load_public: No such file or directory
debug1: identity file /home/fred/.ssh/id_dsa type -1
debug1: key_load_public: No such file or directory
debug1: identity file /home/fred/.ssh/id_dsa-cert type -1
debug1: key_load_public: No such file or directory
debug1: identity file /home/fred/.ssh/id_ecdsa type -1
debug1: key_load_public: No such file or directory
debug1: identity file /home/fred/.ssh/id_ecdsa-cert type -1
debug1: key_load_public: No such file or directory
debug1: identity file /home/fred/.ssh/id_ed25519 type -1
debug1: key_load_public: No such file or directory
debug1: identity file /home/fred/.ssh/id_ed25519-cert type -1
debug1: Enabling compatibility mode for protocol 2.0
debug1: Local version string SSH-2.0-OpenSSH_6.8
debug1: Remote protocol version 2.0, remote software version OpenSSH_6.7
debug1: match: OpenSSH_6.7 pat OpenSSH* compat 0x04000000
debug1: SSH2_MSG_KEXINIT sent
debug1: SSH2_MSG_KEXINIT received
debug1: kex: server->client aes128-ctr umac-64-etm@openssh.com none
debug1: kex: client->server aes128-ctr umac-64-etm@openssh.com none
debug1: expecting SSH2_MSG_KEX_ECDH_REPLY
debug1: Server host key: ecdsa-sha2-nistp256
 SHA256:CEXGTmrVgeY1qEiwFe2Yy3XqrWdjm98jKmX0LK5mlQg
debug1: Host '198.51.100.20' is known and matches the ECDSA host key.
debug1: Found key in /home/fred/.ssh/known_hosts:2
debug1: SSH2_MSG_NEWKEYS sent
debug1: expecting SSH2_MSG_NEWKEYS
debug1: SSH2_MSG_NEWKEYS received
debug1: Roaming not allowed by server
debug1: SSH2_MSG_SERVICE_REQUEST sent
debug1: SSH2_MSG_SERVICE_ACCEPT received
debug1: Authentications that can continue:
 publickey,password,keyboard-interactive
debug1: Next authentication method: publickey
debug1: Trying private key: /home/fred/.ssh/id_rsa
debug1: Trying private key: /home/fred/.ssh/id_dsa
debug1: Trying private key: /home/fred/.ssh/id_ecdsa
debug1: Trying private key: /home/fred/.ssh/id_ed25519
debug1: Next authentication method: keyboard-interactive
debug1: Authentications that can continue:
```

15 http://man.openbsd.org/ssh.1

```
 publickey,password,keyboard-interactive
debug1: Next authentication method: password
debug1: Authentication succeeded (password).
Authenticated to server.example.org ([198.51.100.20]:22).
debug1: channel 0: new [client-session]
debug1: Requesting no-more-sessions@openssh.com
debug1: Entering interactive session.
debug1: client_input_global_request: rtype hostkeys-00@openssh.com want_reply 0
debug1: client_input_channel_req: channel 0 rtype exit-status reply 0
debug1: client_input_channel_req: channel 0 rtype eow@openssh.com reply 0
debug1: channel 0: free: client-session, nchannels 1
debug1: fd 2 clearing O_NONBLOCK
Last login: Sat Mar 14 21:31:33 2015 from 192.0.2.111

...
```

The same login with the maximum of verbosity, **-vvv**, gives around 150 lines of debugging information. Remember that debugging information is sent to **stderr** rather than **stdout** . This will only capture the session in a file, debugging info goes only to the screen, not to the output log:

```
$ ssh -vvv -l fred  somehost.example.org  | tee ~/ssh-output.log
```

The tool tee(1)[16] is like a T-pipe and sends output two directions, one to **stdout** and one to a file.

The following will capture both the debugging info and the session text:

```
$ ssh -vvv -l fred  somehost.example.org  2>&1  | tee ~/ssh-output.log
```

11.2.1 Capturing Client Debugging Information Separately

Regular pipes and redirects work only with **stdout** so that if **-E** is not used to capture debugging output the output on **stderr** must instead be sent to stdout if one is going to capture it at the same time as the actual session. That is done with an extra redirect, **2**>&**1** to capture **stderr**. Mind the spaces, or lack of them.

11.2.2 Changing Client Debugging Levels At Runtime or On The Fly

At run time, when establishing a new connection, just use the **-v** option.

```
$ sftp -v -o "IdentityFile=~/.ssh/weblog.key_rsa" fred@server.example.org
```

The debugging verbosity on the client can be increased just like on the server.

```
$ sftp -vvv -o "IdentityFile=~/.ssh/weblog.key_rsa" fred@server.example.org
```

The extra information can be useful to see exactly what is being sent to or requested of the server.

16 http://man.openbsd.org/tee.1

After the fact, once a connection is established, the escape sequences **~v** and **~V** can be used to increase or decrease the verbosity on the fly. Through them it is possible to change the verbosity of the client in an established connection. When increasing, the client raises its log level to VERBOSE, DEBUG, DEBUG2, and DEBUG3, in that order, if starting from the default of INFO. Conversely, when lowering the log level, the client will descend through ERROR, FATAL, to QUIET if starting from the default of INFO.

11.3 Debugging and Troubleshooting

The server logs are your best friend when troubleshooting. It may be necessary to turn up the log level there temporarily to get more information. It is then also necessary to turn them back to normal after things are fixed to avoid privacy problems or excessively use of disk space.

For example, the SFTP-subsystem logging defaults to ERROR, reporting only errors. To track transactions made by the client, change the log level to INFO or VERBOSE:

```
Subsystem internal-sftp  -l INFO
```

Caution. Again, operating with elevated logging levels would violate the privacy of users, in addition to filling a lot of disk space, and should generally not be used in production once the changes are figured out. Elevated log messages should really be sent to a separate log file during the time they are collected.

By default, some systems send only the normal messages to the regular system log files and ignore the elevated messages. Some save all the messages by default. If the elevated system log messages are not showing up in any of the system logs, the former may be the reason. Either way, check the system log configuration and make sure that the extra messages are only sent to a separate log file and not mixed with the regular system logs. Change the configuration if necessary. This helps keep the logs tidy as well as protect privacy. The system log settings are found in the system log daemon's configuration file, the exact name of which will vary depending on what is installed, but common ones are syslog.conf(5)[17] and rsyslog.conf(5). Notice that this machine configuration below has the more detailed DEBUG messages for the AUTH facility going to a separate log file from the regular AUTH messages:

```
$ grep '^auth\.' /etc/syslog.conf
auth.info                                               /var/log/authlog
auth.debug                                              /var/log/authdebug
```

See syslog(3)[18] for the log facilities and log levels. It is best to limit the time the debugging information is collected and to actively watch while it is collected. However, if it is running for any length of time, and especially if it is left unattended even for a short while, be sure to remeber to add the special log file to the log rotation schedule so that it cannot fill up the partition.

17 https://man.openbsd.org/syslog.conf.5
18 https://man.openbsd.org/syslog.3

Match blocks can help further by setting log levels for specific situations and avoid a situation where everything is logged intensely.

Also, the manual pages for OpenSSH are very well written and many times problems can be solved by finding the right section within the right manual page. At the very minimum, it is important to skim through the four main manual pages for both the programs and their configuration and become familiar with at least the section headings.

- ssh(1)[19]
- ssh_config(5)[20]
- sshd(8)[21]
- sshd_config(5)[22]

Then once the right section is found in the manual page, go over it in detail and become familiar with its contents. The same goes for the other OpenSSH manual pages, depending on the activity. Be sure to use the version of OpenSSH available for your system and the corresponding manual pages, preferably those that are installed on your system to avoid a mismatch. In some cases, the client and the server will be of different versions, so the manual pages for each must be looked up separately. It is also a good idea to review OpenSSH's release notes when a new version is published.

With a few exceptions below, specific examples of troubleshooting are usually given in the cookbook section relevant to a particular activity. So, for example, sorting problems with authentication keys is done in the section on Public Key Authentication[23] itself.

11.3.1 Debugging a script, configuration or key that uses sudo(8)

Usually log levels only need to be changed when writing and testing a script, a new configuration, some new keys, or all three at once. When working with **sudo(8)**, it is especially important to see exactly what the client is sending so as to enter the right pattern into /**etc**/**sudoers** for safety. Using the lowest level of verbosity, the exact string being sent by the client to the remote server is shown in the debugging output:

```
$ rsync -e "ssh -v -i /home/webmaint/.ssh/bkup_key -l webmaint" \
        -a server.example.org:/var/www/ var/backup/www/
...
debug1: Authentication succeeded (publickey).
Authenticated to server.example.org ([192.0.2.20]:22).
debug1: channel 0: new [client-session]
debug1: Requesting no-more-sessions@openssh.com
debug1: Entering interactive session.
debug1: Sending command: rsync --server --sender -vlogDtpre.if . /var/www/
receiving incremental file list
...
```

What **sudoers** then needs is something like the following, assuming account 'webmaint' is in the group 'webmasters':

19 http://man.openbsd.org/ssh.1
20 http://man.openbsd.org/ssh_config.5
21 http://man.openbsd.org/sshd.5
22 http://man.openbsd.org/sshd_config.5
23 https://en.wikibooks.org/wiki/OpenSSH/Cookbook/Authentication_Keys

```
%webmasters ALL=(ALL) NOPASSWD: /usr/local/bin/rsync --server \
--sender -vlogDtpre.if . /var/www/
```

The same method can be used to debug new server configurations or key logins. Once things are set to run as needed, the log level settings can be lowered back to INFO for sshd(8)[24] and to ERROR for **internal-sftp**. Additionally, once the script is left to run in fully automated mode, the client logging information can be set use the syslog(3)[25] system module instead of **stderr** by setting the **-y** option when it is launched.

11.3.2 Debugging a server configuration

Running the server in debug mode provides a lot of information about the connection and a smaller amount about the server configuration. The server's debugging level (**-d**) can be raised once, twice (**-dd**) or thrice (**-ddd**).

```
$ /usr/sbin/sshd -d
```

Note that the server in this case does not detach and become a daemon, so it will terminate when the SSH connection terminates. The server must be started again in order to make a subsequent connection from the client. Though in some ways this is a hassle, it does make sure that session data is a unique set and not mixes of multiple sessions and thus possibly different configurations. Alternately, another option (**-e**) when debugging sends the debugging data to **stderr** to keep the system logs clean.

In recent versions of OpenSSH, it is also possible to log the debug data from the system logs directly to a separate file and keep noise out of the system logs. Since OpenSSH 6.3, the option **-E** will append the debug data to a particular log file instead of sending it to the system log. This facilitates debugging live systems without cluttering the system logs.

```
$ /usr/sbin/sshd -E /home/fred/sshd.debug.log
```

On older versions of OpenSSH, if you need to save output to a file while still viewing it live on the screen, you can use **tee(1)**.

```
$ /usr/sbin/sshd -ddd 2>&1 | tee /tmp/foo
```

That will save output to the file *foo* by capturing what sshd(8)[26] sent to **stderr**. This works with older versions of OpenSSH, but the **-E** option above is preferable.

If the server is remote and it is important to reduce the risk of getting locked out, the experiments on the configuration file can be done with a second instance of sshd(8)[27] using a separate configuration file and listening to a high port until the settings have been tested.

```
$ /usr/sbin/sshd -dd -p 22222 -f /home/fred/sshd_config.test
```

24 http://man.openbsd.org/sshd.8
25 http://man.openbsd.org/syslog.3
26 http://man.openbsd.org/sshd.8
27 http://man.openbsd.org/sshd.8

It is possible to make an extended test (**-T**) of the configuration file. If there is a syntax error, it will be reported, but remember that even sound configurations can still lock you out. The extended test mode can be used by itself, but it is also possible to specify particular connection parameters to use with **-C**. sshd(8)[28] will then process the configuration file in light of the parameters passed to it and output the results. Of particular use, the results of **Match** directives will be shown. So the **-T** option can be supplemented with the **-C** option to show precisely which configuration will be used for various connections.

When passing specific connection parameters to sshd(8)[29] for evaluation, *user*, *host*, and *addr* are the minimum required for extended testing. The following will print out the configurations that will be applied if the user *fred* tries to log in to the host *server.example.org* from the address *192.0.2.15*:

```
$ /usr/sbin/sshd -T -C user=fred,host=server.example.org,addr=192.0.2.15
```

Two more parameters, *laddr* and *lport*, may also be passed. They refer to the server's IP number and port connected to.

```
$ /usr/sbin/sshd -T -C
 user=fred,host=server.example.org,addr=192.0.2.15,laddr=192.0.2.2,lport=2222
```

Those five variables should be able to describe any possible incoming connection.

11.3.3 Debugging a client configuration

Sometimes when debugging a server configuration it is necessary to track the client, too. Since OpenSSH 6.8, the -G option makes ssh(1)[30] print its configuration after evaluating **Host** and **Match** blocks and then exit. That allows viewing of the exact configuration options that will actually be used by the client for a particular connection.

```
$ ssh -G -l fred server.example.org
```

Client configuration is determined in three ways. The first is by run-time options, then by the account's own configuration file, or lastly the system-wide client configuration file. The priority is in that order and whichever value is found first gets used. With sftp(1)[31] the options are also passed to ssh(1)[32].

Invalid or Outdated Ciphers or MACs

A proper client will show the details of the failure. For a bad Message Authentication Code (MAC), a proper client might show something like the following when trying to foist a bad MAC like hmac-md5-96 onto the server:

28 http://man.openbsd.org/sshd.8
29 http://man.openbsd.org/sshd.8
30 http://man.openbsd.org/ssh.1
31 http://man.openbsd.org/sftp.1
32 http://man.openbsd.org/ssh.1

```
no matching mac found: client hmac-md5-96 server umac-64-etm@openssh.com,umac-12
8-etm@openssh.com,hmac-sha2-256-etm@openssh.com,hmac-sha2-512-etm@openssh.com,hma
c-sha1-etm@openssh.co
m,umac-64@openssh.com,umac-128@openssh.com,hmac-sha2-256,hmac-sha2-512,hmac-sha1
```

And for a bad cipher, a proper client might show something like this when trying to foist an arcfour cipher on the server:

```
no matching cipher found: client arcfour server chacha20-poly1305@openssh.co
m,aes128-ctr,aes192-ctr,aes256-ctr,aes128-gcm@openssh.com,aes256-gcm@openssh.com
```

Sometimes when troubleshooting a problem with the client it is necessary to turn to the server logs. In OpenSSH 6.7 unsafe MACs were removed[33] and in OpenSSH 7.2 unsafe ciphers were removed[34], but some third-party clients may still try to use them to establish a connection. In that case, the client might not provide much information beyond a vague message that the server unexpectedly closed the network connection. The server logs will, however, show what happened:

```
fatal: no matching mac found: client hmac-sha1,hmac-sha1-96,hmac-md5 server hmac
-sha2-512-etm@openssh.com,hmac-sha2-256-etm
@openssh.com,umac-128-etm@openssh.com,hmac-sha2-512,hmac-sha2-256,hmac-ripemd160
 [preauth]
```

More recent versions would show a simpler error for a bad MAC.

```
fatal: Unable to negotiate with 192.0.2.37 port 55044: no matching MAC found.
 Their offer: hmac-md5-96 [preauth]
```

A bad cipher would be reported like this:

```
fatal: Unable to negotiate with 192.0.2.37 port 55046: no matching cipher found.
 Their offer: arcfour [preauth]
```

The error message in the server log might not say which MACs or ciphers are actually available. For those, the extended test mode can be used to show the server settings and, in particular, the MACs or ciphers allowed. In its most basic usage the extended test mode would just be -T, as in `/usr/sbin/sshd -T | grep -E 'cipher|macs'` with no other options. For more details and options, see the previous section on "Debugging a server configuration" above.

One solution there is to upgrade the client to one that can handle the right ciphers and MACs. Another option is to switch to a different client, one that can handle the modern ciphers or MACs.

11.3.4 Debugging Key-Based Authentication

The most common causes of failure for public key authentication seem to be for either of two reasons:

33 http://www.openssh.com/txt/release-6.7
34 http://www.openssh.com/txt/release-7.2

- Mangling the public key on the way to getting it into **authorized_keys** on the server
- Incorrect permissions for the files and directories involved, ether on the client or the server. These are the directory for the keys, usually ~/**.ssh**/, or its parent directories, or the **authorized_keys** file, or the private key itself.

As of the time of this writing, it looks like pretty much every failure of key-based authentication described on mailing lists and forums is solved by addressing either or both of those two situations. So, when encountering the error message "Permission denied (publickey,keyboard-interactive)", or similar, see the section on Public Key Authentication[35]. Then see the manual page for sshd(8)[36] and its section on authorized keys. Usually, though not always, it will be obvious when the private key's permissions are incorrect:

```
$ ssh -i ~/.ssh/fred-193.ed25519 fred@192.0.2.193
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@         WARNING: UNPROTECTED PRIVATE KEY FILE!          @
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
Permissions 0664 for '/home/fred/.ssh/fred-193.ed25519' are too open.
It is required that your private key files are NOT accessible by others.
This private key will be ignored.
Load key "/home/fred/.ssh/fred-193.ed25519": bad permissions
```

In addition mangled public keys in the authorized keys file and to incorrect permissions, a very rare third case is if the public and private key files don't match and are not from the same key pair. As mentioned in the section on Public Key Authentication[37], the public and private keys need to match and be part of the same key pair. That is because even before the SSH client uses private key cryptographically, it looks at the file name of the proposed private key and then sends the public key matching that same name, if it exists. If the public key that is on the client does not match the public key on the server in **authorized_keys** then the connection will be denied with the error "Permission denied (publickey,keyboard-interactive)" or similar. That alone is a very good reason to give unique descriptive file names. Note that as mentioned above there are usually other causes to that same error message besides having mismanaged the public key file on the client machine.

The file names for both parts of each key pair have to be kept organized so that the contents match. As for a solution, the way out in the long term is to more carefully manage the keys and their file names. It is solved on the short term by deleting the offending public key file or using the private key to regenerate a new one, overwriting the offending one. Again, this is an unusual edge case and not a common cause of that error.

SSH Too Many Authentication Failures

When there are multiple keys in the authentication agent, the client will try them against the server in an unpredictable order. If the client happens to cycle through enough of the wrong keys first and hits the server's **MaxAuthTries** limit before finding the right key, the server will naturally break off the connection with an error message about too many authentication failures:

35 https://en.wikibooks.org/wiki/OpenSSH/Cookbook/Public_Key_Authentication
36 http://man.openbsd.org/sshd.8
37 https://en.wikibooks.org/wiki/OpenSSH/Cookbook/Public_Key_Authentication

```
"Received disconnect from 203.0.113.110 port 22:2: Too many authentication
 failures
Authentication failed."
```

With increased verbosity, the keys tested and rejected will also be shown:

```
$ ssh -v 203.0.113.110
...
debug1: SSH2_MSG_SERVICE_ACCEPT received
debug1: Authentications that can continue: publickey,keyboard-interactive
debug1: Next authentication method: publickey
debug1: Offering RSA public key: /home/fred/.ssh/key.06.rsa
debug1: Authentications that can continue: publickey,keyboard-interactive
debug1: Offering RSA public key: /home/fred/.ssh/key.02.rsa
debug1: Authentications that can continue: publickey,keyboard-interactive
debug1: Offering RSA public key: /home/fred/.ssh/key.03.rsa
debug1: Authentications that can continue: publickey,keyboard-interactive
debug1: Offering RSA public key: /home/fred/.ssh/key.04.rsa
debug1: Authentications that can continue: publickey,keyboard-interactive
debug1: Offering RSA public key: /home/fred/.ssh/key.01.rsa
debug1: Authentications that can continue: publickey,keyboard-interactive
debug1: Offering RSA public key: /home/fred/.ssh/key.05.rsa
Received disconnect from 203.0.113.110 port 22:2: Too many authentication
 failures
Authentication failed.
```

Each key in the agent gets an annotation which says whether or not the key file was supplied by the user, either in the configuration file or as a run-time argument. The client prefers keys that were specified in the configuration and are also currently in the agent. Then it will try try them in the order in which they were supplied. [37]

There are two solutions if you see the ”Too many authentication failures” error:

One way around this error is to remove keys from the agent one at a time using ssh-add(1)[38] with the **-d** option until there is just the right key left. Refer to each key by its file system path, for example: `ssh-add -d ~/.ssh/some.key.rsa` Because the private key to be removed is looked up in the agent based on the corresponding public key both files must exist. Without matching a public key file, the private key cannot be removed individually from the authentication agent. Instead the whole lot may be removed all at once using the **-D** option. However, is not always practical to do either when many remote systems are used frequently and the agent needs to be kept well-stocked. This is probably not the most practical way.

Another way around this error, and probably the most practical method, is to limit the client to trying only a specific key using the **IdentitiesOnly** configuration directive in conjunction with the **IdentityFile** configuration directive. The latter points explicitly to the right key. Both can be added either as run-time options or in the client's configuration file. As a run-time option, they can be used like this:

```
$ ssh -o IdentitiesOnly=yes -i ~/.ssh/server14.example.org.rsa -l fred
 server14.example.org
```

38 `http://man.openbsd.org/ssh-add.1`

Or these two options could be added to the client configuration file in something like the following way instead.

```
Host server14 server14.example.org
        HostName server14.example.org
        IdentitiesOnly yes
        IdentityFile /home/fred/.ssh/server14.example.org.rsa
        User fred
```

In that way, the server could be reached with either the short name or the fully qualified domain name, whatever names are listed under the **Host** directive.

```
$ ssh server14
```

Remember that options are selected from the client configuration file on a first-match basis. Because the first-match wins, specific rules must come before more general rules.

Signing Failed for ... Agent Refused Operation

As mentioned, incorrect permissions on client files or directories are a common cause of authentication failure when attempting key-based authentication. Not all the errors are clear about that, however. Here is an example of a misleading error message which is actually caused by incorrect permissions:

```
$ ssh -i ~/.ssh/key-ed25519 fred@server.example.org
sign_and_send_pubkey: signing failed for ED25519 "/home/fred/.ssh/key-ed25519"
 from agent: agent refused operation
fred@server.example.org: Permission denied (publickey).
```

The solution for that is to ensure that no other accounts can read or write to the private key, nor should others be able to write to the **.ssh** directory or its parent.

11.3.5 Debugging Chrooted SFTP Accounts

The most common problem seems to be bad directory permissions. The chroot directory, and all directories above it, must be owned by root and not writable by any other user or group. Even though these directories' group memberships do not have to be root, if any of them is not root then it must not be group writable either. Failure to use the correct ownership will result in not being able to log in with the affected accounts. The errors when login is attempted will look like this from the client side:

```
$ sftp fred@192.0.2.206
fred@192.02.206's password:
packet_write_wait: Connection to 192.0.2.206: Broken pipe
Couldn't read packet: Connection reset by peer
```

The error message is much clearer on the server side:

```
Aug  4 23:52:38 server sshd[7075]: fatal: bad ownership or modes for chroot
 directory component "/home/fred/"
```

Check the directory permissions for the chroot target and all directories above it. If even one is off it must be fixed so that it is owned by root and not writable by any others. There are many, many routes to get there. Here are two was to set up chroot permissions:

- One quick way to fix the permissions is to change both ownership and group membership of the directory to root. Same for all directories above the chroot target.

```
$ ls -lhd /home/ /home/fred/
drwxr-xr-x 3 root  root  4.0K Aug  4 20:47 /home/
drwxr-xr-x 8 root  root  4.0K Aug  4 20:47 /home/fred/
```

That will work with the **ChrootDirectory** directive set to **%h** but has some drawbacks that will quickly become obvious when adding files or directories.

- Another easy way to fix the permissions is to change both the account's home directory and the **ChrootDirectory** directive. Arrange the account's home directory so that it is under a unique directory owned by root, such as the user name itself:

```
$ ls -lhd /home/ /home/fred/ /home/fred/fred/
drwxr-xr-x 3 root  root  4.0K Aug  4 20:47 /home/
drwxr-x--- 3 root  fred  4.0K Aug  4 20:47 /home/fred/
drwxr-x--- 8 fred  fred  4.0K Aug  4 20:47 /home/fred/fred/
```

Then chroot the account to the parent directory and combine that with an alternate starting directory working from the user name token with the **-d** option for the SFTP server.

```
ChrootDirectory /home/%u
ForceCommand internal-sftp -d %u
```

Then when the account connects it will see only its own directory and no other parts of the system.

11.3.6 Debugging RC Scripts Interfering with SFTP Sessions

The SFTP connection will drop if there are any extraneous data either direction on **stdin** , from the client or the server. A common mistake in that area is if /**etc**/**ssh**/**sshrc** or ~/**.ssh**/**rc** send anything at all to **stdout** instead of being quiet. There the output, which would be **stdout** on the server, is received by the client on **stdin**, but matches no correct protocol and thus causes the client to disconnect. So, even in the case of using the RC scripts, the response from the server must remain 8-bit clean or an error will occur:

```
$ sftp server.example.org
Received message too long 1400204832
```

That one message will be the main clue. Increasing the verbosity of the SFTP client with **-v** won't provide more relevant information.

Also, the standard logs on the server will only show that the client disconnected and not provide any information why. At higher levels of logging, some extraneous reads and corresponding discards might be noticed but that is all. Below is a log sample recorded at the verbosity DEBUG3 showing such an example.

```
...
debug2: subsystem request for sftp by user fred
debug1: subsystem: exec() /usr/libexec/sftp-server
Starting session: subsystem 'sftp' for fred from 198.51.100.38 port 37446 id 0
...
debug2: channel 0: read 13 from efd 12
debug3: channel 0: discard efd
debug2: channel 0: read 12 from efd 12
debug3: channel 0: discard efd
debug2: channel 0: read 15 from efd 12
debug3: channel 0: discard efd
debug2: channel 0: read 18 from efd 12
debug3: channel 0: discard efd
...
```

Again, neither RC script is allowed produce any output on **stdout** during use of SFTP or it will ruin the connection. If an RC script does produce output, it must be redirected to a system log, to a file, or sent to **stderr** instead of **stdout**. Regular interactive SSH connections are not disturbed by use of **stdout** and the client will just display whatever is sent. See the manual page for sshd(8)[39] in the section "SSHRC" for more.

The same restriction goes for any other part of the SSH service which runs over **stdin** and **stdout**, such as **ProxyJump** or some uses of **ProxyCommand**. So another example of potential interference would be when using **LocalCommand** with the client to specify a command to execute on the local machine after successfully connecting to the server. Any output from it also needs to be redirected to **stderr**. If **LocalCommand** ends up interfering with **ProxyJump** then the connection will appear to hang at the stage when **stdout** gets used.

11.3.7 Debugging When An SSH Agent Has The Correct Private Key But Does Not Use It

In older versions of OpenSSH, the public key must also be available on the client at the time the private key is loaded into the agent. If it is not then without the matching public key the agent will not be able to use a private key unless other arrangements are made. A symptom of this is that while specifying the key via a run-time argument works, the same key does not work via the agent.

Upgrading to a more recent version of OpenSSH is a better option. Otherwise a work-around is to specify the private key either as a run-time argument to the client or in the ssh_config(5)[40] file, from there the client will find the correspondingly named public key file. Importantly the client will still use the key in the agent yet use the designated matching public key file, so the private key file does not have to contain anything at all and could even be empty.

```
$ ssh -i some_key_ed25519 fred@server.example.org
```

39 http://man.openbsd.org/sshd.8
40 http://man.openbsd.org/ssh_config.5

However, if it is undesirable to have the private key accessible on the file system or if the private key is only in the agent and not itself available via the file system, then the public key can be specified directly instead.

```
$ ssh -i some_key_ed25519.pub fred@server.example.org
```

Either way, another way is to name the key in ssh_config(5)[41] instead using the **IdentityFile** configuration directive. If the file with the public key is missing, it can be regenerated from the private key using ssh-keygen(1)[42] with the **-y** option.

11.4 SSH Client Error Messages And Common Causes

As rehash of the above, below are some client-side error messages with some of the more common reasons[38] for those messages. Neither list of errors nor the reasons for the errors are comprehensive. The suggestions for causes just touch on one or two of the commonly seen reasons. There is no substitute for checking the actual logs, especially on the server. The server log files are usually **/var/log/auth.log** on most system or a variant of that name. Occasionally you will find the information in the file **/var/log/secure** instead.

No address associated with name:

```
$ ssh nonesuch.example.org
ssh: Could not resolve hostname nonesuch.example.org: no address associated with
 name
```

The destination's host name does not exist in DNS. Was it spelled correctly?

Operation timed out:

```
$ ssh 198.51.100.89
ssh: connect to host 198.51.100.89 port 22: Operation timed out
```

There's not a system associated with that IP address, or else a packet filter is causing trouble.

Connection timed out:

```
$ ssh 198.51.100.89
ssh: connect to host 198.51.100.89 port 22: Connection timed out
```

You can't get there from here. It is probable that the destination machine is disconnected from the network or that the network which it is on is not reachable.

Connection refused:

```
$ ssh www.example.org
ssh: connect to host www.example.org port 22: Connection refused
```

41 http://man.openbsd.org/ssh_config.5
42 http://man.openbsd.org/ssh-keygen.1

The destination system exists but there is no SSH service available at the port being tried. Is the destination right? It might be the wrong system, SSH might not be running at all, SSH might be listening on another port, or a packet filter might be blocking connections.

Permission denied:

```
$ ssh www.example.org
fred@www.example.org: Permission denied (publickey,keyboard-interactive).
```

That is usually a sign of a problem cause by the wrong user name, wrong authentication method, wrong SSH key or SSH certificate, or wrong file permissions in the authorized keys file on the destination system. If it is a matter of SSH key based authentication, see the chapter on Public Key Authentication[43] for more thorough coverage. If it is a question of SSH certificates, see the corresponding Certificate-based Authentication[44] chapter. It can also be a matter of server-side settings with **AllowGroups**, **DenyGroups**, or similar configuration directives at the destination. Any of those possibilities can really only be identified and solved by checking the log output from sshd(8)[45].

No matching host key type found:

```
$ ssh www.example.org
Unable to negotiate with 198.51.100.89 port 22: no matching host key type found.
 Their offer: ssh-rsa,ssh-dss
```

The SSH daemon on that server is really outdated. Contact the system administrator about an upgrade and get them moving. More details can be seen with the **-v** option on the client.

```
$ ssh -v fred@example.org
...
debug1: SSH2_MSG_KEXINIT received
debug1: kex: algorithm: diffie-hellman-group-exchange-sha256
debug1: kex: host key algorithm: (no match)
Unable to negotiate with 198.51.100.89 port 22: no matching host key type found.
 Their offer: ssh-rsa,ssh-dss
```

In the above shell session, it is the server which is badly in need of an update as shown by the deprecated key exchange algorithms which it tries to use. Again, for emphasis, the solution is to update the outdated software. If a specific, old version of an particular operating system absolutely must be used for a while longer, see about getting a back port of the SSH daemon. Many GNU/Linux distros even have specific back port repositories.

If you would like to see which key exchange algorithms the client supports try using the **-Q** option.

```
$ ssh -Q kex
diffie-hellman-group1-sha1
diffie-hellman-group14-sha1
diffie-hellman-group14-sha256
diffie-hellman-group16-sha512
diffie-hellman-group18-sha512
diffie-hellman-group-exchange-sha1
```

43 https://en.wikibooks.org/wiki/OpenSSH/Cookbook/Public_Key_Authentication
44 https://en.wikibooks.org/wiki/OpenSSH/Cookbook/Certificate-based_Authentication
45 http://man.openbsd.org/sshd.8

```
diffie-hellman-group-exchange-sha256
ecdh-sha2-nistp256
ecdh-sha2-nistp384
ecdh-sha2-nistp521
curve25519-sha256
curve25519-sha256@libssh.org
sntrup761x25519-sha512@openssh.com
```

12 Development

It is possible to advance OpenSSH through donations of hardware or money. See the OpenSSH project web site at www.openssh.com[1] for details.

OpenSSH is a volunteer project with the goal of making quality software. In that way it relies upon hardware and cash donations to keep the project rolling. Funds are needed for daily operation to cover network line subscriptions and electrical costs. If two dollars were given for every download of the OpenSSH source code from the master site in 2015, ignoring the mirrors, or if a penny was donated for every instance of PF or OpenSSH installed with a mainstream operating system or phone in 2015[39], then funding goals for the year would be met. Hardware is needed for development and porting to new architectures and platforms always requires new hardware.

OpenSSH is currently developed by two teams. The first team works to provide code that is as clean, simple and secure as possible. It is part of the OpenBSD project. The second team works using this core version and ports it to a great many other operating systems. Thus there are two development tracks, the OpenBSD core and the portable version. All the work is done in countries that permit export of cryptography.

12.1 Use the Source, Luke

The main development branch of OpenSSH is part of the OpenBSD project. So the source code for the "-current" branch of OpenBSD is where to look for latest activity. Nightly, bleeding-edge snapshots of OpenSSH itself are thus publicly available from OpenBSD's CVS tree[2]. Use a mirror when possible.

The source code for the portable releases of OpenSSH are published using anonymous Git, so no password is needed to download source from the read-only repository. The repository is provided and maintained by Damien Miller.

```
git://anongit.mindrot.org/openssh.git
```

We ask anyone wishing to report security bugs in OpenSSH to please use the contact address given in the source and to practice responsible disclosure.

1 http://www.openssh.com
2 https://www.openbsd.org/anoncvs.html

12.2 libssh

libssh is an independent project that provides a multiplatform C library implementing the SSHv2 and SSHv1 protocols for client and server implementations. With libssh, developers can remotely execute programs, transfer files and use a secure and transparent tunnel for your remote applications.

libssh is available under LGPL 2.1 license, on the web page [3]

Features:

- Key Exchange Methods: curve25519-sha256@libssh.org, ecdh-sha2-nistp256, diffie-hellman-group1-sha1, diffie-hellman-group14-sha1
- Hostkey Types: ecdsa-sha2-nistp256, ssh-dss, ssh-rsa
- Ciphers: aes256-ctr, aes192-ctr, aes128-ctr, aes256-cbc, aes192-cbc, aes128-cbc, 3des-cbc, des-cbc-ssh1, blowfish-cbc
- Compression Schemes: zlib, zlib@openssh.com, none
- MAC hashes: hmac-sha1, none
- Authentication: none, password, public-key, hostbased, keyboard-interactive, gssapi-with-mic
- Channels: shell, exec (incl. SCP wrapper), direct-tcpip, subsystem, auth-agent-req@openssh.com
- Global Requests: tcpip-forward, forwarded-tcpip
- Channel Requests: x11, pty, exit-status, signal, exit-signal, keepalive@openssh.com, auth-agent-req@openssh.com
- Subsystems: sftp(version 3), publickey(version 2), OpenSSH Extensions
- SFTP: statvfs@openssh.com, fstatvfs@openssh.com
- Thread-safe: Just don't share sessions
- Non-blocking: it can be used both blocking and non-blocking
- Your sockets: the app hands over the socket, or uses libssh sockets
- OpenSSL or gcrypt: builds with either

Additional Features:

- Client and server support
- SSHv2 and SSHv1 protocol support
- Supports Linux, UNIX, BSD, Solaris, OS/2 and Windows
- Full API documentation and a tutorial
- Automated test cases with nightly tests
- Event model based on poll(2), or a poll(2)-emulation.

12.3 libssh2

libssh2 is another independent project providing a lean C library implementing the SSH2 protocol for embedding specific SSH capabilities into other tools. It has a stable, well-

3 `https://www.libssh.org/`

documented API for working on the client side with the different SSH subsystems: Session, Userauth, Channel, SFTP, and Public Key. The API can be set to either blocking or non-blocking. The code uses strict name spaces, is C89-compatible and builds using regular GNU Autotools.

libssh2 is available under a modified BSD license. The functions are each documented in their own manual pages. The project web site contains the documentation, source code and examples: [4]

There is a mailing list for **libssh2** in addition to an IRC channel. The project is small, low-key and, as true to the spirit of the Internet, a meritocracy. Hundreds of specific functions allow specific activities and components to be cherry-picked and added to an application:

- Shell and SFTP sessions
- Port forwarding
- Password, public-key, host-based keys, and keyboard-interactive authentication methods.
- Key Exchange Methods diffie-hellman-group1-sha1, diffie-hellman-group14-sha1, diffie-hellman-group-exchange-sha1
- Host Key Types: ssh-rsa and ssh-dss
- Ciphers: aes256-ctr, aes192-ctr, aes128-ctr, aes256-cbc (rijndael-cbc@lysator.liu.se), aes192-cbc, aes128-cbc, 3des-cbc, blowfish-cbc, cast128-cbc, arcfour, arcfour128, or without a cipher.
- Compression Scheme zlib or without compression
- Message Authentication Code (MAC) algorithms for hashes: hmac-sha1, hmac-sha1-96, hmac-md5, hmac-md5-96, hmac-ripemd160 (hmac-ripemd160@openssh.com), or none at all
- Channels: Shell, Exec – including the SCP wrapper, direct TCP/IP, subsystem
  - Channel Requests: x11, pty
- Subsystems: sftp version 3, public-key version 2
- Thread-safe, blocking or non-blocking API
- Your sockets: the app hands over the socket, calls select() etc.
- Builds with either OpenSSL or gcrypt

See also the library **libcurl** which supports SFTP and SCP URLs.

12.4 Thrussh

Thrussh is an SSH library written in Rust and available under the Apache License version 2.0. It is a full implementation of the SSH 2 protocol. The only non-Rust part is the crypto back end, which uses ring[5] instead. It is designed to work on any platform and to use asynchronous I/O. The project web site contains the documentation, source code, and examples. The code is accessible using **darcs**:

```
darcs get https://pijul.org/thrussh
```

4 http://www.libssh2.org/
5 https://github.com/briansmith/ring

It is not an implementation of an actual server or client, but instead contains all the elements needed to write custom clients and servers using Rust.

12.5 Other language bindings for the SSH protocols

What follows is a list of additional independent resources by programming language:

12.5.1 Perl

- Net::SSH2[6]: a wrapper module for **libssh2**.
- Net::SSH::Perl[7]: a full SSH/SFTP implementation in pure Perl. Unfortunately this module is not being maintained any more and has several open bugs. Also, installing it can be a daunting task due to some of its dependencies.
- Net::OpenSSH[8]: a wrapper for OpenSSH binaries and other handy programs (**scp**, **rsync** , **sshfs**). It uses OpenSSH multiplexing feature in order to reuse connections.
- Net::OpenSSH::Parallel[9] a module build on top of **Net::OpenSSH** that allows to transfer files and run programs on several machines in parallel efficiently.
- SSH::Batch[10] another module build on top of **Net::OpenSSH** that allows to run programs on several hosts in parallel.
- Net::SSH::Expect[11]: this module uses Expect[12] to drive interactive shell sessions run on top of SSH.
- Net::SSH[13]: a simple wrapper around any SSH client. It does not support password authentication and is very slow as it establishes a new SSH connection for every remote program invoked.
- Net::SCP[14] and Net::SCP::Expect[15]: modules wrapping the **scp** program. Note that **Net::SSH2**, **Net::SSH::Perl** and **Net::OpenSSH** already support file transfers via **scp** natively.
- Net::SFTP::Foreign[16]: a full SFTP client written in Perl with lots of bells and whistles. By default is uses **ssh** to connect to the remote machines but it can also run on top of **Net::SSH2** and **Net::OpenSSH**.
- GRID::Machine[17], IPC::PerlSSH[18] and SSH::RPC[19]: these modules allow to distribute and run Perl code on remote machines through SSH.

6 https://search.cpan.org/perldoc?Net::SSH2
7 https://search.cpan.org/perldoc?Net::SSH::Perl
8 https://search.cpan.org/perldoc?Net::OpenSSH
9 https://search.cpan.org/perldoc?Net::OpenSSH::Parallel
10 https://search.cpan.org/perldoc?SSH::Batch
11 https://search.cpan.org/perldoc?Net::SSH::Expect
12 https://search.cpan.org/search?query=expect
13 https://search.cpan.org/perldoc?Net::SSH
14 https://search.cpan.org/perldoc?Net::SCP
15 https://search.cpan.org/perldoc?Net::SCP::Expect
16 https://search.cpan.org/perldoc?Net::SFTP::Foreign
17 https://search.cpan.org/perldoc?GRID::Machine
18 https://search.cpan.org/perldoc?IPC::PerlSSH
19 https://search.cpan.org/perldoc?SSH::RPC

12.5.2 Python

Paramiko[20]

- [21]

Fabric[22]

- [23]

libssh2[24]

- [25]

TwistedConch[26]

- [27]

12.5.3 Ruby

Net::SSH[28]

- [29]

Capistrano[30]

- [31]

12.5.4 Java

Jaramiko[32]

- [33]

JSch[34] - a pure Java implementation of SSH2.

- [35]

20 http://www.lag.net/paramiko/
21 http://www.lag.net/paramiko/
22 http://docs.fabfile.org/
23 http://docs.fabfile.org/
24 http://www.no-ack.org/2010/11/python-bindings-for-libssh2.html
25 http://www.no-ack.org/2010/11/python-bindings-for-libssh2.html
26 https://twistedmatrix.com/trac/wiki/TwistedConch
27 https://twistedmatrix.com/trac/wiki/TwistedConch
28 https://github.com/net-ssh
29 https://github.com/net-ssh
30 https://github.com/capistrano/capistrano
31 https://github.com/capistrano/capistrano
32 http://www.lag.net/jaramiko/
33 http://www.lag.net/jaramiko/
34 http://www.jcraft.com/jsch/
35 http://www.jcraft.com/jsch/

13 Cookbook

14 Remote Processes

One of the main functions of OpenSSH is that of accessing and running programs on other systems. That is, after all, one of the main purposes of the program. There are several ways to expand upon that, either interactively or as part of unattended scripts. So in addition to an interactive login, ssh(1)[1] can be used to simply execute a program or script. Logout is automatic when the program or script has run its course. Some combinations are readily obvious. Others require more careful planning. Sometimes it is enough of a clue just to know that something can be done, at other times more detail is required. A number of examples of useful combinations of using OpenSSH to run remote tasks follow.

14.1 Run a Remote Process

An obvious use of ssh(1)[2] is to run a program on the remote system and then exit. Often this is a shell, but it can be any program available to the account. For feedback, ssh(1)[3] passes whatever exit value was returned by the remote process. When a remote process is completed, ssh(1)[4] will terminate and pass on the exit value of the last remote process to complete. So in this way it can be used in scripts and the outcome of the remote processes can be used by the client system.

The following will run true(1)[5] on the remote system and return success, exit code 0, to the local system where ssh(1)[6] was run.

```
$ ssh -l fred server.example.org /bin/true
$ echo $?
```

The following will run false(1)[7] on the remote system and return failure, exit code 1, to the local system where ssh(1)[8] was run.

```
$ ssh -l fred server.example.org /bin/false
$ echo $?
```

1 http://man.openbsd.org/ssh.1
2 http://man.openbsd.org/ssh.1
3 http://man.openbsd.org/ssh.1
4 http://man.openbsd.org/ssh.1
5 http://man.openbsd.org/true.1
6 http://man.openbsd.org/ssh.1
7 http://man.openbsd.org/false.1
8 http://man.openbsd.org/ssh.1

If any other values, from 0 to 255, were returned, ssh(1)[9] will pass them back the local host from the remote host.

14.1.1 Run a Remote Process and Capture Output Locally

Output from programs run on the remote machine can be saved locally using a normal redirect. Here we run dmesg(8)[10] on the remote machine:

```
$ ssh -l fred server.example.org dmesg > dmesg.from.server.log
```

Interactive processes will be difficult or impossible to operate in that manner because no output will be seen. For interactive processes requiring any user input, output can be piped through tee(1)[11] instead to send the output both to the file and to **stdout**. This example runs an anonymous FTP session remotely and logs the output locally.

```
$ ssh -l fred server.example.org "ftp -a anotherserver" | tee ftp.log
```

It may be necessary to force pseudo-TTY allocation to get both input and output to be properly visible.

```
$ ssh -t -l fred server.example.org "ftp -a anotherserver" | tee
 /home/fred/ftp.log
```

The simplest way to read data on one machine and process it on another is to use pipes.

```
$ ssh fred@server.example.org 'cat /etc/ntpd.conf' | diff /etc/ntpd.conf -
```

14.1.2 Run a Local Process and Capture Remote Data

Data can be produced on one system and used on another. This is different than tunneling X, where both the program and the data reside on the other machine and only the graphical interface is displayed locally. Again, the simplest way to read data on one machine and use it on another is to use pipes.

```
$ cat /etc/ntpd.conf | ssh fred@server.example.org 'diff /etc/ntpd.conf -'
```

In the case where the local program expects to read a file from the remote machine, a named pipe can be used in conjunction with a redirect to transfer the data. In the following example, a named pipe is created as a transfer point for the data. Then ssh(1)[12] is used to launch a remote process which sends output to **stdout** which is captured by a redirection on the local machine and sent to the named pipe so a local program can access the data via the named pipe.

9 http://man.openbsd.org/ssh.1
10 http://man.openbsd.org/dmesg.8
11 http://man.openbsd.org/tee.1
12 http://man.openbsd.org/ssh.1

In this particular example, it is important to add a filter rule to tcpdump(8)[13] itself to prevent an infinite feedback loop if ssh(1)[14] is connecting over the same interface as the data being collected. This loop is prevented by excluding either the SSH port, the host used by the SSH connection, or the corresponding network interface.

```
$ mkfifo -m 600 netdata

$ ssh -fq -i /home/fred/.ssh/key_rsa \
        'sudo tcpdump -lqi eth0 -w - "not port 22"' > netdata

$ wireshark -k -i netdata &
```

Any **sudo(8)** privileges for tcpdump(8)[15] also need to operate without an interactive password, so great care and precision must be exercised to spell out in /**etc**/**sudoers** exactly which program and parameters are to be permitted and nothing more. The authentication for ssh(1)[16] must also occur non-interactively, such as with a key and key agent. Once the configurations are set, ssh(1)[17] is run and sent to the background after connecting. With ssh(1)[18] in the background the local application is launched, in this case wireshark(1)[19], a graphical network analyzer, which is set to read the named pipe as input.

On some systems, process substitution can be used to simplify the transfer of data between the two machines. Doing process substitution requires only a single line.

```
$ wireshark -k -i <( ssh -fq -i /home/fred/.ssh/key_rsa \
        'sudo tcpdump -lqi eth0 -w - "not port 22"' )
```

However, process substitution is not POSIX compliant and thus not portable across platforms. It is limited to bash(1)[20] only and not present in other shells. So, for portability, use a named pipe.

14.2 Run a Remote Process While Either Connected or Disconnected

There are several different ways to leave a process running on the remote machine. If the intent is to come back to the process and check on it periodically then a terminal multiplexer is probably the best choice. For simpler needs there are other approaches.

14.2.1 Run a Remote Process in the Background While Disconnected

Many routine tasks can be set in motion and then left to complete on their own without needing to stay logged in. When running remote process in background it is useful to spawn a shell just for that task.

13 http://man.openbsd.org/tcpdump.8
14 http://man.openbsd.org/ssh.1
15 http://man.openbsd.org/tcpdump.8
16 http://man.openbsd.org/ssh.1
17 http://man.openbsd.org/ssh.1
18 http://man.openbsd.org/ssh.1
19 http://linux.die.net/man/1/wireshark
20 http://linux.die.net/man/1/bash

```
$ ssh -t -l fred server.example.org 'sh -c "tar zcf /backup/usr.tgz /usr/" &'
```

Another way is to use a terminal multiplexer. An advantage with them is being able to reconnect and follow the progress from time to time, or simply to resume work in progress when a connection is interrupted such as when traveling. Here tmux(1)[21] reattaches to an existing session or else, if there is none, then creates a new one.

```
$ ssh -t -l fred server.example.org "tmux a -d || tmux"
```

On older systems, screen(1)[22] is often available. Here it is launched remotely to create a new session if one does not exists, or re-attach to a session if one is already running. So if no screen(1)[23] session is running, one will be created.

```
$ ssh -t -l fred server.example.org "screen -d -R"
```

Once a screen(1)[24] session is running, it is possible to detach it and close the SSH connection without disturbing the background processes it may be running. That can be particularly useful when hosting certain game servers on a remote machine. Then the terminal session can then be reattached in progress with the same two options.

```
$ ssh -t -l fred server.example.org "screen -d -R"
```

There is more on using terminal multiplexers tmux(1)[25] or on below. In some environments it might be necessary to also use pagsh(1)[26], especially with Kerberos, see below. Or nohup(1)[27] might be of use.

Keeping Authentication Tickets for a Remote Process After Disconnecting

Authentication credentials are often deleted upon logout and thus any remaining processes no longer have access to whatever the authentication tokens were used for. In such cases, it is necessary to first create a new credential cache sandbox to run an independent process in before disconnecting.

```
$ pagsh
$ /usr/local/bin/a-slow-script.sh
```

Kerberos and AFS are two examples of services that require valid, active tickets. Using pagsh(1)[28] is one solution for those environments.

21 http://man.openbsd.org/tmux.1
22 http://linux.die.net/man/1/screen
23 http://linux.die.net/man/1/screen
24 http://linux.die.net/man/1/screen
25 http://man.openbsd.org/tmux.1
26 http://linux.die.net/man/1/pagsh
27 http://man.openbsd.org/nohup.1
28 http://linux.die.net/man/1/pagsh

14.2.2 Automatically Reconnect and Restore an SSH Session Using tmux(1) or screen(1)

Active, running sessions can be restored after either an intentional or accidental break by using a terminal multiplexer. Here ssh(1)[29] is to assume that the connection is broken after 15 seconds (three tries of five seconds each) of not being able to reach the server and to exit. Then the tmux(1)[30] session is reattached or, if absent, created.

```
$ while ! ssh -t fred@example.org -o 'ServerAliveInterval 5' \
    'tmux attach -d || tmux new-session';
$ do true;
$ done
```

Then each time ssh(1)[31] exits, the shell tries to connect with it again and when that happens to look for a tmux(1)[32] session to attach to. That way if the TCP or SSH connections are broken, none of the applications or sessions stop running inside the terminal multiplexer. Here is an example for screen(1)[33] on older systems.

```
$ while ! ssh -t fred@example.org -o 'ServerAliveInterval 5' \
    'screen -d -R;'
$ do true;
$ done
```

The above examples give only an overly simplistic demonstration where at their most basic they are useful to resume a shell where it was after the TCP connection was broken. Both tmux(1)[34] and screen(1)[35] are quite capable of much more and worth exploring especially for travelers and telecommuters.

See also the section on "Public Key Authentication[36]" to integrate keys into the process of automatically reconnecting.

14.2.3 Sharing a Remote Shell

Teaching, team programming, supervision, and creating documentation are some examples of when it can be useful for two people to share a shell. There are several options for read-only viewing as well as for multiple parties being able to read and write.

Read-only Monitoring or Logging

Pipes and redirects are a quick way to save output from an SSH session or to allow additional users to follow along read-only.

One sample use-case is when a router needs to be reconfigured and is available via serial console. Say the router is down and a consultant must log in via another user's laptop's

29 http://man.openbsd.org/ssh.1
30 http://man.openbsd.org/tmux.1
31 http://man.openbsd.org/ssh.1
32 http://man.openbsd.org/tmux.1
33 http://linux.die.net/man/1/screen
34 http://man.openbsd.org/tmux.1
35 http://linux.die.net/man/1/screen
36 https://en.wikibooks.org/wiki/OpenSSH/Cookbook/Public_Key_Authentication

connection to access the router's serial console and it is necessary to supervise what is done or help at certain stages. It is also very useful in documenting various activities, including configuration or installation.

Read-only Using tee(1)

Capture shell activity to a log file and optionally use tail to watch it real time. The utility tee(1)[37], like a t-joint in plumbing, is used here to send output to two destinations, both **stdout** and a file.

```
$ ssh fred@server.example.org | tee /tmp/session.log
```

The resulting file can be monitored live in another terminal using tail(1)[38] or after the fact with a pager like less(1)[39].

Force Serial Session with Remote Logging Using tee(1)

The tee(1)[40] utility can capture output from any program that can write to **stdout**. It is very useful for walking someone at a remote site through a process, supervising, or building documentation.

This example uses chroot(8)[41] to keep the choice of actions as limited as possible. Actually building the chroot jail is a separate task. Once built, the guest user is made a member of the group 'consult'. The serial connection for the test is on device ttyUSB0, which is a USB to serial converter and cu(1)[42] is used for the connection. tee(1)[43] takes the output from cu(1)[44] and saves a copy to a file for logging while the program is used. The following is would go in sshd_config(5)[45]

```
Match Group consult
        ChrootDirectory /var/chroot-test
        AllowTCPForwarding no
        X11Forwarding no
        ForceCommand cu -s 19200 -l /dev/ttyUSB0 | tee /var/tmp/cu.log
```

With that one or more people can follow the activity in cu(1)[46] as it happens using tail(1)[47] by pointing it at the log file on the remote server.

```
$ tail -f /var/tmp/cu.log
```

37 http://man.openbsd.org/tee.1
38 http://man.openbsd.org/tail.1
39 http://man.openbsd.org/less.1
40 http://man.openbsd.org/tee.1
41 http://man.openbsd.org/chroot.8
42 http://man.openbsd.org/cu.1
43 http://man.openbsd.org/tee.1
44 http://man.openbsd.org/cu.1
45 http://man.openbsd.org/sshd_config.5
46 http://man.openbsd.org/cu.1
47 http://man.openbsd.org/tail.1

Or the log can be edited and used for documentation. It is also possible for advanced users of tmux(1)[48] or screen(1)[49] to allow read-only observers.

Scripting

It is possible to automate some of the connection. Make a script, such as **/usr/local/bin/screeners**, then use that script with the **ForceCommand** directive. Here is an example of a script that tries to reconnect to an existing session. If no sessions already exist, then a new one is created and automatically establishes a connection to a serial device.

```
#!/bin/sh

# try attaching to an existing screen session,
# or if none exist, make a new screen session

/usr/bin/screen -d -R || \
        /usr/bin/screen \
            /bin/sh -c "/usr/bin/cu -s 19200 -l /dev/ttyUSB0 | \
            /usr/bin/tee /tmp/consultant.log"
```

Interactive Sharing Using a Terminal Multiplexer

The terminal multiplexers tmux(1)[50] or screen(1)[51] can attach two or more people to the same session.[40] The session can either be read-only for some or read-write for all participants.

tmux(1)

If the same account is going to be sharing the session, then it's rather easy. In the first terminal, start tmux(1)[52] where 'sessionname' is the session name:

```
$ tmux new-session -s sessionname
```

Then in the second terminal:

```
$ tmux attach-session -t sessionname
```

That's all that's needed if the same account is logged in from different locations and will share a session. For different users, you have to set the permissions on the tmux(1)[53] socket so that both users can read and write it. That will first require a group which has both users as members.

48 http://man.openbsd.org/tmux.1
49 http://linux.die.net/man/1/screen
50 http://man.openbsd.org/tmux.1
51 http://linux.die.net/man/1/screen
52 http://man.openbsd.org/tmux.1
53 http://man.openbsd.org/tmux.1

Then after both accounts are in the shared group, in the first terminal, the one with the main account, start tmux(1)[54] as before but also assign a name for the session's socket. Here 'sessionname' is the session name and 'sharedsocket' is the name of the socket:

```
$ tmux -S /tmp/shareddir/sharedsocket new-session -s sessionname
```

Then change the group of the socket and the socket's directory to a group that both users share in common. Make sure that the socket permissions allow the group to write the socket. In this example the shared group is 'foo' and the socket is /**tmp**/**shareddir**/**sharedsocket** .

```
$ chgrp foo /tmp/shareddir/
$ chgrp foo /tmp/shareddir/sharedsocket
$ chmod u=rwx,g=rx,o= /tmp/shareddir/
$ chmod u=rw,g=rw,o=  /tmp/shareddir/sharedsocket
```

Finally, have the second account log in attach to the designated session using the shared socket.

```
$ tmux -S /tmp/shareddir/sharedsocket attach-session -t sessionname
```

At that point, either account will be able to both read and write to the same session.

screen(1)

If the same account is going to share a screen(1)[55] session, then it's an easy procedure. In the one terminal, start a new session and assign a name to it. In this example, 'sessionname' is the name of the session:

```
$ screen -S sessionname
```

In the other terminal, attach to that session:

```
$ screen -x sessionname
```

If two different accounts are going to share the same screen(1)[56] session, then the following extra steps are necessary. The first user does this when initiating the session:

```
$ screen -S sessionname
^A :multiuser on
^A :acladd user2
```

Then the second user does this:

```
$ screen -x user1/sessionname
```

54 http://man.openbsd.org/tmux.1
55 http://linux.die.net/man/1/screen
56 http://linux.die.net/man/1/screen

In screen(1)[57], if more than one user account is used the `aclchg` command can remove write access for the other user: `^A :aclchg user -w "#"`. Note that screen(1)[58] must run as SUID for multiuser support. If it is not set, you will get an error message reminding you when trying to connect the second user. You might also have to set permissions for /**var**/**run**/**screen** to 755.

14.3 Display Remote Graphical Programs Locally Using X11 Forwarding

It is possible to run graphical programs on the remote machine and have them displayed locally by forwarding X11, the current implementation of the X Window system. X11 is used to provide the graphical interface on many systems. See the website www.X.org[59] for its history and technical details. It is built into most desktop operating systems. It is even distributed as part of Macintosh OS X, though there it is not the default method of graphical display.

X11 forwarding is off by default and must be enabled on both the SSH client and server if it is to be used.

X11 also uses a client server architecture and the X server is the part that does the actual display for the end user while the various programs act as the clients and to the server. Thus by putting the client and server on different machines and forwarding the X11 connections, it is possible to run programs on other computers but have them displayed and available as if they were on the user's computer.

A note of caution is warranted. Allowing the remote machine to forward X11 connections will allow it and its applications to access many devices and resources on the machine hosting the X server. Regardless of the intent of the users, these are the devices and other resources accessible to the user account. So forwarding should only be done when the other machine, that user account, and its applications are reliable.

On the server side, to enable X11 forwarding by default, put the line below in sshd_config(5)[60], either in the main block or a **Match** block:

```
X11Forwarding yes
```

On the client side, forwarding of X11 is also off by default, but can be enabled using three different ways. It can be enabled in ssh_config(5)[61] or else using either the **-X** or **-Y** run-time arguments.

```
$ ssh -l fred -X desk.example.org
```

The connection will be slow, however. If responsiveness is a factor, it may be relevant to consider a SOCKS proxy instead or some other technology all together like FreeNX.

57 http://linux.die.net/man/1/screen
58 http://linux.die.net/man/1/screen
59 http://www.x.org/
60 http://man.openbsd.org/sshd_config.5
61 http://man.openbsd.org/ssh_config.5

14.3.1 Using ssh_config(5) to Specify X11 Forwarding

X11 forwarding can be enabled in **/etc/ssh_config** for all accounts for all outgoing SSH connections or for just specific hosts by configuring ssh_config(5)[62].

```
X11Forwarding yes
```

It is possible to apply ssh_config(5)[63] settings to just one account in **~/.ssh/config** to limit forwarding by default to an individual host by hostname or IP number.

```
Host desk.example.org
        X11Forwarding yes
```

And here it is enabled for a specific machine by IP number

```
Host 192.168.111.25
        X11Forwarding yes
```

Likewise, use limited pattern matching to allow forwarding for a subdomain or a range of IP addresses. Here it is enabled for any host in the *pool.example.org* domain, any host from 192.168.100.100 to 192.168.100.109, and any host from 192.168.123.1 through 192.168.123.254:

```
Host *.pool.example.org
        X11Forwarding yes

Host 192.168.100.10?
        X11Forwarding yes

Host 192.168.123.*
        X11Forwarding yes
```

Again, X11 is built-in to most desktop systems. There is an optional add-on for OS X which has its roots in NextStep. X11 support may be missing from some particularly outdated, legacy platforms, however. But even there it is often possible to retrofit them using the right tools, one example being the tool Xming[64].

14.4 Unprivileged sshd(8) Service

It is possible to run sshd(8)[65] on a high port using an unprivileged account. It will not be able to fork to other accounts, so login will be only possible on the same account which is running the unprivileged service.

There are three general steps to running an unprivileged SSH service on a normal, unprivileged account.

1) Prior to launching the SSH daemon under an unprivileged account, some unprivileged host keys must be created. These keys will be used to identify this service to connecting

62 http://man.openbsd.org/ssh_config.5
63 http://man.openbsd.org/ssh_config.5
64 http://www.straightrunning.com/XmingNotes/
65 http://man.openbsd.org/sshd.8

clients, especially on return visits. Their creation only needs to be done once per account and the passphrase must be left empty.

```
$ ssh-keygen -q -t ed25519 -N '' \
        -f /home/fred/.ssh/ssh_host_key_ed25519 \
        -C 'unprivileged SSH host key for fred'
```

If needed, do likewise for ECDSA and RSA keys. Be sure that that the permissions are correct. The directory which the keys are in and its parents should not be writable by other accounts and the private SSH host key itself must not be readable by any other accounts.

2) When the keys are in place, test them by starting the SSH server manually. One ways is by using the **-h** option to point to these alternative host key files. Note that because the account is unprivileged only the ports from 1024 on up are available, in this example port 2222 is set using the **-p** option.

```
$ /usr/sbin/sshd -D \
        -h /home/fred/.ssh/ssh_host_key_ed25519 \
        -h /home/fred/.ssh/ssh_host_key_ecdsa -p 2222
```

Verify that the new service is accessible from another system. Especially make sure that any packet filters, if they are any along the way, are set to pass the right port. Incoming connections, and any packet filters, are going to have to take the alternative port into consideration in normal uses. Another approach would be to add the unprivileged SSH service as an onion service. See the chapter on Proxies and Jump Hosts[66] about that.

Note that the above examples use the default configuration file for the daemon. If a different configuration file is needed for special settings, then add the **-f** option to the formula.

```
$ /usr/sbin/sshd -D -f /home/fred/.ssh/sshd_config
```

Specific configuration directives can be set for the unprivileged account by using an alternative configuration file. Unprivileged host keys can be identified in that file along with a designated alternative listening port and many other customizations. It might also be helpful to log the unprivileged service separately from any other SSH services already on the same system by changing the **SyslogFacility** directive to something other than AUTH which is the default.

3) Automate the startup, if that is a goal.

The **-D** option keeps the process from detaching and becoming a daemon. During a test that might be useful, but later in actual use it might not be, it depends on how the process is launched.

14.4.1 What Unprivileged sshd(8) Processes Look Like

Because an unprivileged account is being used, privilege separation will not occur and all the child process will run in the same account as the original sshd(8)[67] process. Here's how

66 https://en.wikibooks.org/wiki/OpenSSH/Cookbook/Proxies_and_Jump_Hosts#Providing_SSH_as_an_Onion_Service

67 http://man.openbsd.org/sshd.8

that can look while listening on port 2222 for an incoming connection using an unprivileged account, using the default configuration file with a few overrides:

```
$ pgrep -d , sshd | COLUMNS=200 xargs ps -w -o user,pid,args -p
USER         PID COMMAND
fred     1997992 sshd: /usr/sbin/sshd -h /home/fred/.ssh/ssh_host_key_ed25519 -h
 /home/fred/.ssh/ssh_host_key_ecdsa -p 2222 [listener] 0 of 10-100 startups
```

Then once someone has connected but not finished logging in yet, it can look like this. A monitor is forked but even though it is labeled [priv] it is not privileged, and instead still running in the same unprivileged account as the parent process. It in turn forks a child, here process 1998150, to manage the authentication:

```
$ pgrep -d , sshd | COLUMNS=200 xargs ps -w -o user,pid,args -p
USER         PID COMMAND
fred     1997992 sshd: /usr/sbin/sshd -h /home/fred/.ssh/ssh_host_key_ed25519 -h
 /home/fred/.ssh/ssh_host_key_ecdsa -p 2222 [listener] 1 of 10-100 startups
fred     1998149 sshd: fred [priv]
fred     1998150 sshd: fred [net]
```

Finally, once login has been successful, that unprivileged process is dropped and a new unprivileged process spun up to manage the interactive session.

```
$ pgrep -d , sshd | COLUMNS=200 xargs ps -w -o user,pid,args -p
USER         PID COMMAND
fred     1997992 sshd: /usr/sbin/sshd -h /home/fred/.ssh/ssh_host_key_ed25519 -h
 /home/fred/.ssh/ssh_host_key_ecdsa -p 2222 [listener] 0 of 10-100 startups
fred     1998149 sshd: fred [priv]
fred     1998410 sshd: fred@pts/20
```

Above, process 1998410 is managing the interactive session.

14.5 Locking Down a Restricted Shell

A restricted shell sets up a more controlled environment than what is normally provided by a standard interactive shell. Though it behaves almost identically to a standard shell, it has many exceptions regarding capabilities that are whitelisted, leaving the others disabled. The restrictions include, but are not limited to, the following:

- The SHELL, ENV, and PATH variables cannot be changed.
- Programs can't be run with absolute or relative paths.
- Redirections that create files can't be used (specifically >, >|, >>, <>).

Common high-end shells like bash(1)[68], ksh(1)[69], and zsh(1)[70] all can be launched in restricted mode. See the manual pages for the individual shells for the specifics of how to invoke restrictions.

Even with said restrictions, there are several ways by which it is trivial to escape from a restricted shell: If normal shells are available anywhere in the path, they can be launched instead. If regular programs in the available path provide shell escapes to full shells, they

68 https://linux.die.net/man/1/bash
69 http://man.openbsd.org/man1/ksh.1
70 https://linux.die.net/man/1/zsh

too can be used. Finally, if sshd(8)[71] is configured to allow arbitrary programs to be run independently of the shell, a full shell can be launched instead. So there's more to safely using restricted shells than just setting the account's shell to `/bin/rbash` and calling it a day. Several steps are needed to make that as difficult as possible to escape the restrictions, especially over SSH.

(The following steps assume familiarity with the appropriate system administration tools and their use. Their selection and use are not covered here.)

First, create a directory containing a handful of symbolic links that point to white-listed programs. The links point to the programs that the account should be able to run when the directory is added to the PATH environment variable. These programs should have no shell escape capabilities and, obviously, they should not themselves be unrestricted shells.

If you want to prevent exploration of the system at large, remember to also lock the user into a chroot or jail. Even without programs like ls(1) and cat(1), exploration is still possible (see below: "ways to explore without ls(1) and cat(1)").

Symbolic links are used because the originals are, hopefully, maintained by package management software and should not be moved. Hard links cannot be used if the original and whitelisted directories are in separate filesystems. Hard links are necessary if you set up a chroot or jail that excludes the originals.

```
$ ls -l /usr/local/rbin/
total 8
lrwxr-xr-x  1 root  wheel    22 Jan 17 23:08 angband -> /usr/local/bin/angband
lrwxr-xr-x  1 root  wheel     9 Jan 17 23:08 date -> /bin/date
-rwxr-xr-x  1 root  wheel  2370 Jan 17 23:18 help
lrwxr-xr-x  1 root  wheel    12 Jan 17 23:07 man -> /usr/bin/man
lrwxr-xr-x  1 root  wheel    13 Jan 17 23:09 more -> /usr/bin/more
lrwxr-xr-x  1 root  wheel    28 Jan 17 23:09 nethack ->
 /usr/local/bin/nethack-3.4.3
...
```

Next, create a minimal **.profile** for that account. Set its owner to 'root'. Do this for its parent directory too (which is the user's home directory). Then, allow the account's own group to read both the file and the directory.

```
$ cd /home/fred

$ cat .profile
PATH=/usr/games:/usr/local/rbin
export PATH HOME TERM

$ ls -ld . .profile
drwxr-xr-x  3 root  fred    512 Jan 17 23:20 .
-rw-r--r--  1 root  fred     48 Jan 17 23:20 .profile
```

Next, create a group for the locked down account(s) and populate it. Here the account is in the group *games* and will be restricted through its membership in that group.

```
$ groups fred
fred games
```

71 `http://man.openbsd.org/man8/sshd.8`

Next, lock down SSH access for that group or account through use of a **ForceCommand** directive in the server configuration and apply it to the selected group. This is necessary to prevent trivial circumvention through the SSH client by calling a shell directly, such as with `ssh -t fred@server.example.org /bin/sh` or similar. Remember to disable forwarding if it is not needed. For example, the following can be appended to sshd_config(5)[72] so that any account in the group 'games' gets a restricted shell no matter what they try with the SSH client.

```
Match Group games
        X11Forwarding no
        AllowTcpForwarding no
        ForceCommand rksh -l
```

Note that the restricted shell is invoked with the **-l** option by the **ForceCommand** so that it will be a login shell that reads and executes the contents of /**etc**/**profile** and **$HOME**/**.profile** if they exist and are readable. This is necessary to set the custom PATH environment variable. Again, be sure that **$HOME**/**.profile** is not in any way editable or overwritable by the restricted account. Also note that this disables SFTP access by that account, which prevents quite a bit of additional mischief.

Last but not least, set the account's login shell to the restricted shell. Include the full path to the restricted shell. It might also be necessary to add it to the list of approved shells found in /**etc**/**shells** first.

14.6 Beware: ways to explore without ls(1) and cat(1)

```
# To see a list of files in the current working directory:
$ echo *

# To see the contents of a text file:
$ while read j; do echo "$j"; done  <.profile
```

72 http://man.openbsd.org/man5/sshd_config.5

15 Tunnels

In tunneling, or port forwarding, a local port is connected to a port on a remote host or vice versa. So connections to the port on one machine are in effect connections to a port on the other machine.

The ssh(1)[1] options **-f** (go to background), **-N** (do not execute a remote program) and **-T** (disable pseudo-tty allocation) can be useful for connections that are used only for creation of tunnels.

15.1 Tunneling

In regular port forwarding, connections to a local port are forwarded to a port on a remote machine. This is a way of securing an insecure protocol or of making a remote service appear as local. Here we forwarded VNC in two steps. First make the tunnel:

```
$ ssh -L 5901:localhost:5901 -l fred desktop.example.org
```

In that way connections on the local machine made to the forwarded port will in effect be connecting to the remote machine.

Multiple tunnels can be specified at the same time. The tunnels can be of any kind, not just regular forwarding. See the next section below for reverse tunnels. For dynamic forwarding see the section Proxies and Jump Hosts[2].

```
$ ssh -L 5901:localhost:5901 \
      -L 5432:localhost:5432 \
      -l fred desktop.example.org
```

If a connection is only used to create a tunnel, then it can be told not to execute any remote programs (**-N**), making it a non-interactive session, and also to drop to the background (**-f**).

```
$ ssh -fN -L 3128:localhost:3128 -l fred server.example.org
```

1 http://man.openbsd.org/ssh.1
2 https://en.wikibooks.org/wiki/OpenSSH/Cookbook/Proxies_and_Jump_Hosts

Note that **-N** will work even if the **authorized_keys** forces a program using the **command="..."** option. So a connection using **-N** will stay open instead of running a program and then exiting.

The three connections above could be saved in the SSH client's configuration file, **~/.ssh/config** and even given shortcuts.

```
Host desktop desktop.example.org
        HostName desktop.example.org
        User fred
        LocalForward 5901 localhost:5901

Host postgres
        HostName desktop.example.org
        User fred
        LocalForward 5901 localhost:5901
        LocalForward 5432 localhost:5432

Host server server.example.org
        HostName server.example.org
        User fred
        ExitOnForwardFailure no
        LocalForward 3128 localhost:3128

Host *
        ExitOnForwardFailure yes
```

With those settings, the tunnels listed are added automatically when connecting to *desktop*, *desktop.example.org*, *postgres*, *server*, or *server.example.org*. The catchall configuration at the end applies to any of the above hosts which have not already set **ExitOnForwardFailure** to 'no' and the client will refuse to connect if a tunnel cannot be made. The first obtained value for any given configuration directive will be used, but the file's contents can be overidden with run-time options passed on the command line.

15.1.1 Tunneling Via A Single Intermediate Host

Tunneling can go via one intermediate host to reach a second host, and the latter does not need to be on a publicly accessible network. However, the target port on the second remote machine does have to be accessible on the same network as the first. Here, *192.168.0.101* and *bastion.example.org* must be on the same network and, in addition, *bastion.example.org* has to be directly accessible to the client machine running **ssh(1)**. So, port 80 on *192.168.0.101* has to be available to the machine *bastion.example.org*.

```
$ ssh -fN -L 1880:192.168.0.101:80 -l fred bastion.example.org
```

Thus, once the tunnel is made, to connect to port 80 on *192.168.2.101* via the host *bastion.example.org*, connect to port 1880 on *localhost*. This way works for one or two hosts. It is also possible to chain multiple hosts, using different methods.

Securing a Hop, Tunneling Via One Or More Intermediate Hosts

Here, the idea is to limit the ability of a group of users to the bare minimum needed to pass through a jump host yet still be able to forward ports onward to other machines. If the account is sufficiently locked down then the bastion can only be used for forwarding and not shell access, scripts, or even SFTP. The following settings on the bastion host in sshd_config(5)[3] prevent either shell access or SFTP but still allow port forwarding.

```
Match Group tunnelers
        ForceCommand /bin/false
        PasswordAuthentication no
        ChrootDirectory %h
        PermitTTY no
        X11Forwarding no
        AllowTcpForwarding yes
        PermitTunnel no
        Banner none
```

Note that their home directories, but not the files within them, must be owned by root and writable by only root because of the **ChrootDirectory** configuration directive there in sshd_config(5)[4]. Also, because of the **PasswordAuthentication** configuration directive keys will have to be set up in the home directory in **~/.ssh/authorized_keys**, if an alternate location is not already configured.

```
$ ssh -N -L 9980:localhost:80 -J fred@bastion.example.org fred@192.168.79.124
```

In that way port 9980 on the client is directed via *bastion.example.org* through to port 80 on *192.168.79.124*.

In cases where the bastion must have a reverse tunnel from the inner host in order to reach it, then the same method works but with the prerequisite of a reverse tunnel from the inner host to the bastion first.

For more about passing through intermediate computers, see the Cookbook section on Proxies and Jump Hosts[5].

15.1.2 Finding The Process ID (PID) Of A Tunnel Which Is In The Background

When a tunneled connection is sent to the background execution using the **-f** option for the client, there is not currently an automatic way to find the process ID (PID) of the task sent to the background. Background processes are often used for port forwarding or reverse port forwarding. Here is an example of port forwarding, also called tunneling. The connection is made and then the client goes away, leaving the tunnel available in the background, connecting port 2194 on the local host to port 194 of the remote system's local host.

```
$ ssh -Nf -L 2194:localhost:194 fred@203.0.113.214
```

3 http://man.openbsd.org/sshd_config.5
4 http://man.openbsd.org/sshd_config.5
5 https://en.wikibooks.org/wiki/OpenSSH/Cookbook/Proxies_and_Jump_Hosts

The special **$!** variable remains empty, even if **$?** reports success or failure of the action. The reason is that the shell's job control did not put the client into the background. Instead the client runs in the foreground for a moment and then exits normally, after leaving a different process to run in the background via a fork. The process ID of the original client vanishes since that client is gone.

Finding the process ID usually takes at least two steps. Some of the ways to retroactively identify the process involve trying ps(1)[6] and rummaging through the output for all that account's processes, but that is unnecessary effort. If the background SSH client is the most recent one, then pgrep(1)[7] can be used, or else the output needs to be comma delimited and fed into ps(1)[8] via xargs(1)[9] or process substitution.

```
$ ps uw | less

$ pgrep -n -x ssh

$ pgrep -d, -x ssh | xargs ps -p

$ ps -p $(pgrep -d, -x ssh)
```

Some variations on the above might be needed depending on operating system if the **-d** option is not supported.

```
$ pgrep -x ssh | xargs -n 1 ps -o user,pid,ppid,args -p | awk 'NR==1 || $3==1'
USER       PID  PPID COMMAND
fred     97778     1 ssh -fN -L 8008:localhost:80 fred@203.0.113.8
fred     14026     1 ssh -fN -L 8183:localhost:80 fred@203.0.113.183
fred     79522     1 ssh -fN -L 8228:localhost:80 fred@203.0.113.228
fred     49773     1 ssh -fN -L 8205:localhost:80 203.0.113.205
```

Either way, note that all the connections running in the background have a Parent Process ID (PPID) of 1, the process control initialization system for the operating system.

Being aware of this shortcoming, a proactive approach can be used with the **ControlMaster** and **ControlPath** configuration directives in order to leave a socket to read to get the background task's process ID.

```
$ ssh -M -S ~/.ssh/sockets/pid.%C -fN -L 5901:localhost:5901 fred@203.0.113.122

$ ssh -S ~/.ssh/sockets/pid.%C -O check 203.0.113.122
```

The **-M** option causes the client to go into Master mode for multiplexing using the socket designated by the **-S** option. Then, after **-f** forks the client into the background, The control command *check* will then use the socket to check that the master process is running and report the Process ID.

It is a good idea for the socket to be in an isolated directory not readable or writable by other accounts. Aside from the complexity, a noticeable down side is that it can be

6 http://man.openbsd.org/ps.1
7 http://man.openbsd.org/pgrep.1
8 http://man.openbsd.org/ps.1
9 http://man.openbsd.org/xargs.1

possible for the socket to be reused for additional connections. See the Cookbook section on Multiplexing[10] for more about the risks and additional uses.

15.2 Reverse Tunneling

A reverse tunnel forwards connections in the opposite direction of a regular tunnel, that is to say the opposite direction from that which the SSH session is initiated. With remote forwarding as it is also called, an SSH session begins from the local host to a remote host while a port is forwarded from the remote host to one the local host. There are two stages in using reverse tunneling: first connect from endpoint A to endpoint B using SSH with remote forwarding enabled; second connect other systems to the designated port on endpoint B and that port is then forwarded to endpoint A. So while system A initiated the SSH connection to system B, the connections to the designated port on B are sent over to A over the reverse tunnel. Once the SSH connection is made, the reverse tunnel can be used on endpoint B the same as a regular tunnel, even though the endpoint A initiated the SSH connection.

Remote forwarding is method which can be used to forward SSH over SSH in order to work on an otherwise inaccessible system, such as an always-on SBC behind a home router. First, open an SSH session from the inaccessible system to another which is accessible including a designated reverse tunnel. In this example, while the SSH connection is from a local system (endpoint A) to a remote system (endpoint B), that connection contains a reverse tunnel from port 2022 on that remote system (endpoint B) to port 22 over on the local system:

```
$ ssh -fN -R 2022:localhost:22 -l fred server.example.org
```

Lastly, using the example above, a connection is made on the remote machine, *server.example.org*, to the reverse tunnel connection on port 2022. Thus even though the connection is made to port 2022 on *locahost* on *server.example.org*, the packets end up over on port 22 on the system which initiated the SSH initial connection carrying the reverse tunnel.

```
$ ssh -p 2022 -l fred localhost
```

Thus that example allows SSH access to an otherwise inaccessible system via SSH. SSH goes out, makes a reverse tunnel, then the second system can connect at will to the first via SSH as long as the tunnel persists. If keys and a loop are used to generate the SSH connection with the remote forwarding then the reverse tunnel can be maintained automatically.

The next example makes VNC available over SSH from an otherwise inaccessible system via a second system, *server.example.org*. Starting from the system with a running VNC server, reverse forward the port for the first VNC display locally to the third VNC display over on *server.example.org*:

```
$ ssh -fNT -R 5903:localhost:5901 -l fred server.example.org
```

Then on the system *server.example.org*, people can connect to that system's *localhost* address on its third VNC display and be patched through to the originating system:

10 https://en.wikibooks.org/wiki/OpenSSH/Cookbook/Multiplexing

```
$ xvncviewer :3
```

That also is an example of how the forwarded ports don't have to be the same.

Remote forwarding can be included in the SSH client's configuration file using the **RemoteForward** directive. See the next subsection for that.

A common use-case for reverse tunneling is when you have to access a system or service which is behind either NAT or a firewall or both and thus incoming SSH connections are blocked, but you have direct access to a second system outside the firewall which can accept incoming connections. In such cases it is easy to make a reverse tunnel from the internal system behind the firewall to the second system on the outside. Once the SSH connection has made the reverse tunnel, to connect to the internal system from outside, other systems can connect to the forwarded port on the remote system. The remote system on the outside then acts as a relay server to forward connections to the initiating system on the inside.

15.2.1 Configuring Remote Forwarding Using The Client Configuration File

The **RemoteForward** client configuration directive can be used in ssh_config(5)[11] to establish a reverse tunnel from another system. The **ForkAfterAuthentication** would be **-f** and **SessionType** would be **-N** as run time arguments and, of course, optional in this example which is the same as the first example in the subsection above:

```
Host server server.example.org
        User fred
        HostName server.example.org
        ForkAfterAuthentication yes
        SessionType none
        RemoteForward 2022 localhost:22
```

With that in place over on the other system, one can establish a reverse tunnel from *server* by simply connecting to it by entering `ssh server`. Then over on *server.example.org*, connecting to *localhost* on port 2022 will carry through back to the original system on port 22.

```
Host server server.example.org
        User fred
        HostName server.example.org
        ForkAfterAuthentication yes
        SessionType none
        RequestTTY no
        RemoteForward 5903 localhost:5901
```

Likewise that duplicates the second example from the subsection above but in the client configuration file instead of using run time arguments. More about that is covered in the chapter on using The Client Configuration File[12].

11 http://man.openbsd.org/ssh_config
12 https://en.wikibooks.org/wiki/OpenSSH/Cookbook/The_Client_Configuration_File

15.3 Adding or Removing Tunnels within an Established Connection

It is possible to add or remove tunnels, reverse tunnels, and SOCKS proxies to or from an existing connection using escape sequences. The default escape character is the tilde (~) and the full range of options is described in the manual page for ssh(1)[13]. Escape sequences only work if they are the first characters entered on a line and followed by a return. When adding or removing a tunnel to or from an existing connection, **~C**, the command line is used.

To add a tunnel in an active SSH session, use the escape sequence to open a command line in SSH and then enter the parameters for the tunnel:

```
~C
L 2022:localhost:22
```

To remove a tunnel from an active SSH session is almost the same. Instead of -L, -R, or -D we have -KL, -KR, and -KD plus the port number. Use the escape sequence to open a command line in SSH and then enter the parameters for removing the tunnel.

```
~C
KL2022
```

15.3.1 Adding or Removing Tunnels within a Multiplexed Connection

There is an additional option for forwarding when multiplexing. More than one SSH connection can be multiplexed over a single TCP connection. Control commands can be passed to the master process to add or drop port forwarding to the master process.

First a master connection is made and a socket path assigned.

```
$ ssh -S '/home/fred/.ssh/%h:%p' -M server.example.org
```

Then using the socket path, it is possible to add port forwarding.

```
$ ssh -O forward -L 2022:localhost:22 -S '/home/fred/.ssh/%h:%p'
 fred@server.example.org
```

Since OpenSSH 6.0 it is possible to cancel specific port forwarding using a control command.

```
$ ssh -S  "/home/fred/.ssh/%h:%p" -O cancel -L 2022:localhost:22
 fred@server.example.org
```

For more about multiplexing, see the Cookbook section on Multiplexing[14].

13 http://man.openbsd.org/ssh.1

14 https://en.wikibooks.org/wiki/OpenSSH/Cookbook/Multiplexing

15.4 Restricting Tunnels to Specific Ports

By default, port forwarding will allow forwarding to any port if it is allowed at all. The way to restrict which ports a user can use in forwarding is to apply the **PermitOpen** option on the server side either in the server's configuration or inside the user's public key in **authorized_keys**. For example, with this setting in sshd_config(5)[15] all users can forward only to port 7654 on the server, if they try forwarding:

```
PermitOpen localhost:7654
```

Multiple ports may be specified on the same line if separated by whitespace.

```
PermitOpen localhost:7654 localhost:3306
```

If the client tries to forward to a disallowed port, there will be a warning message that includes the text "open failed: administratively prohibited: open failed" while the connection otherwise continues as normal. However, even if the client has **ExitOnForwardFailure** in its configuration a connection will still succeed, despite the warning message.

However, if shell access is available, it is possible to run other port forwarders, so without further restrictions, **PermitOpen** is more of a reminder or recommendation than a restriction. But for many cases, such a reminder might be enough.

For reverse tunnels, the **PermitListen** option is available instead. It determines which port on the remote system is available. So the following, for example, allows `ssh -R 2022:localhost:`*`xxxx`*, where *xxxx* can be any available port number at the origin of the reverse tunnel, but only 2022 on the far end of the tunnel.

```
PermitListen localhost:2022
```

The **PermitOpen** or **PermitListen** options can be used as part of one or more **Match** blocks if forwarding options need to vary depending on various combinations of user, group, client address or network, server address or network, and/or the listening port used by sshd(8)[16]. If using the **Match** criteria to selectively apply rules for port forwarding, it is also possible to prevent the account from getting an interactive shell by setting **PermitTTY** to **no**. That will prevent the allocation of a pseudo-terminal (PTY) on the server thus prevent shell access, but allow other programs to still be run unless an appropriate forced command is specified in addition.

```
Match Group mariadbusers
        PermitOpen localhost:3306
        PermitTTY no
        ForceCommand /usr/bin/true
```

With that stanza in sshd_config(5)[17] it is only possible to connect by adding the **-N** option to avoid executing a remote command.

15 http://man.openbsd.org/sshd_config.5
16 http://man.openbsd.org/sshd.8
17 http://man.openbsd.org/sshd_config.5

```
$ ssh -N -L 3306:localhost:3306 server.example.org
```

The **-N** option can be used alone or with the **-f** option which drops the client to the background once the connection is established.

Without the **ForceCommand** option in a **Match** block in the server configuration, if an attempt is made to get a PTY by a client that is blocked from getting one, the warning "PTY allocation request failed on channel *n*" will show up on the client, with *n* being the channel number, but otherwise the connection will succeed without a remote shell and the port(s) will still be forwarded. Various programs, including shells, can still be specified by the client, they just won't get a PTY. So to really prevent access to the system other than forwarding, a forced command is needed. The tool **true(1)** comes in handy for that. Note that **true(1)** might be in a different location on different systems.

15.4.1 Limiting Port Forwarding Requests Using Keys Only

The following **authorized_keys** line shows the **PermitOpen** option prepended to a key in order to limit a user connecting with that particular key to forwarding to just port 8765:

```
permitopen="localhost:8765" ssh-ed25519 AAAAC3NzaC1lZDI1NT...
```

Multiple **PermitOpen** options may be applied to the same public key if they are separated by commas and thus a key can allow multiple ports.

By default, shell access is allowed. With shell access it is still possible to run other port forwarders. The **no-pty** option inside the key can facilitate making a key that only allows forwarding and not an interactive shell if combined with an appropriate forced command using the **command** option. Here is an example of such a key as it would be listed in **authorized_keys**:

```
no-pty,permitopen="localhost:9876",command="/usr/bin/true" ssh-ed25519
 AAAAC3NzaC1lZDI1NT...
```

The **no-pty** option blocks interactive shell. The client will still connect to the remote server and will allow forwarding but will respond with an error including the message "PTY allocation request failed on channel *n*". But as mentioned in the previous subsection, there are still a lot of ways around that and adding the **command** option hinders them.

This method is awkward to lock down. If the account has write access in any way, directly or indirectly, to the **authorized_keys** file, then it is possible for the user to add a new key or overwrite the existing key(s) with more permissive options. In order to prevent that, the server has to be configured to look for the file in a location outside of the user's reach. See the section on Public Key Authentication[18] for details on that. The methods described above in the previous subsection using sshd_config(5)[19] might be more practical in many cases.

18 https://en.wikibooks.org/wiki/OpenSSH/Cookbook/Public_Key_Authentication
19 http://man.openbsd.org/sshd_config.5

16 Automated Backup

Using OpenSSH with keys can facilitate secure automated backups. rsync(1)[1][41], tar(1)[2], and dump(8)[3] are the foundation for most backup methods. It's a myth that remote root access must be allowed. If root access is needed, sudo(8)[4] works just fine or, in the case of zfs(8)[5], the OpenZFS Delegation System. Remember that until the backup data has been tested and shown to restore reliably it does not count as a backup copy.

16.1 Backup with rsync(1)

rsync(1)[6] is often used to back up both locally and remotely. It is fast and flexible and copies incrementally so only the changes are transferred, thus avoiding wasting time re-copying what is already at the destination. It does that through use of its now famous algorithm. When working remotely, it needs a little help with the encryption and the usual practice is to tunnel it over SSH.

The rsync(1)[7] utility now defaults to using SSH and has since 2004[42]. Thus the following connects over SSH without having to add anything extra:

```
$ rsync -a fred@server.example.org:./archive/ \
    /home/fred/archive/.
```

But use of SSH can still be specified explicitly if additional options must be passed to the SSH client:

```
$ rsync -a -e 'ssh -v' \
    fred@server.example.org:./archive/ \
    /home/fred/archive/.
```

For some types of data, transfer can sometimes be expedited greatly by using rsync(1)[8] with compression, **-z**, if the CPUs on both ends can handle the extra work. However, it can also slow things down. So compression is something which must be tested in place to find out one way or the other whether adding it helps or hinders.

1 http://linux.die.net/man/1/rsync
2 http://man.openbsd.org/tar.1
3 http://man.openbsd.org/dump.8
4 http://linux.die.net/man/8/sudo
5 https://linux.die.net/man/8/zfs
6 http://linux.die.net/man/1/rsync
7 http://linux.die.net/man/1/rsync
8 http://linux.die.net/man/1/rsync

16.1.1 Rsync with Keys

Since rsync(1)[9] uses SSH by default it can even authenticate using SSH keys by using the **-e** option to specify additional options. In that way it is possible to point to a specific SSH key file for the SSH client to use when establishing the connection.

```
$ rsync --exclude '*~' -avv \
    -e 'ssh -i ~/.ssh/key_bkup_rsa' \
    fred@server.example.org:./archive/ \
    /home/fred/archive/.
```

Other configuration options can also be sent to the SSH client in the same way if needed, or via the SSH client's configuration file. Furthermore, if the key is first added to an agent, then the key's passphrase only needs to be entered once. This is easy to do in an interactive session within a modern desktop environment. In an automated script, the agent will have to be set up with explicit socket names passed along to the script and accessed via the **SSH_AUTH_SOCK** variable.

16.1.2 Root Level Access for rsync(1) with sudo(8)

Sometimes the backup process needs access to a different account other than the one which can log in. That other account is often root which for reasons of least privilege is usually denied direct access via SSH. rsync(1)[10] can invoke sudo(8)[11] on the remote machine, if needed.

Say you're backing up from the server to the client. rsync(1)[12] on the client uses ssh(1)[13] to make the connection to rsync(1)[14] on the server. rsync(1)[15] is invoked from client with **-v** passed to the SSH client to see exactly what parameters are being passed to the server. Those details will be needed in order to incorporate them into the server's configuration for sudo(8)[16]. Here the SSH client is run with a single level of increased verbosity in order to show which options must be used:

```
$ rsync \
  -e 'ssh -v \
          -i ~/.ssh/key_bkup_rsa  \
          -t              \
          -l bkupacct'   \
  --rsync-path='sudo rsync' \
  --delete   \
  --archive  \
  --compress \
  --verbose  \
  bkupacct@server:/var/www/ \
  /media/backups/server/backup/
```

9 http://linux.die.net/man/1/rsync
10 http://linux.die.net/man/1/rsync
11 http://linux.die.net/man/8/sudo
12 http://linux.die.net/man/1/rsync
13 http://man.openbsd.org/ssh.1
14 http://linux.die.net/man/1/rsync
15 http://linux.die.net/man/1/rsync
16 http://linux.die.net/man/8/sudo

There the argument –**rsync-path** tells the server what to run in place of rsync(1)[17]. In this case it runs `sudo rsync`. The argument **-e** says which remote shell tool to use. In this case it is ssh(1)[18]. For the SSH client being called by the rsync(1)[19] client, **-i** says specifically which key to use. That is independent of whether or not an authentication agent is used for ssh keys. Having more than one key is a possibility, since it is possible to have different keys for different tasks.

You can find the exact settings(s) to use in /**etc**/**sudoers** by running the SSH in verbose mode (**-v**) on the client. Be careful when working with patterns not to match more than is safe.

Adjusting these settings will most likely be an iterative process. Keep making changes to /**etc**/**sudoers** on the server while watching the verbose output until it works as it should. Ultimately /**etc**/**sudoers** will end up with a line allowing rsync(1)[20] to run with a minimum of options.

Steps for rsync(1) with Remote Use of sudo(8) Over SSH

These examples are based on fetching data from a remote system. That is to say that the data gets copied from /**source**/**directory**/ on the remote system to /**destination**/**directory**/ locally. However, the steps will be the same for the reverse direction, but a few options will be placed differently and –**sender** will be omitted. Either way, copy-paste from the examples below won't work.

Preparation: Create a single purpose account to use only during the backups, create a pair of keys to use only for that account, then make sure you can log in to that account with ssh(1)[21] with and without those keys.

```
$ ssh -i ~/.ssh/key_bkup_ed25519 bkupacct@www.example.org
```

The account on the server is named 'bkupacct' and the private Ed25519 key is **~**/**.ssh**/**key_bkup_ed25519** on the client. On the server, the account 'bkupacct' is a member of the group 'backups'. See the section on Public Key Authentication[22] if necessary.

The public key, **~**/**.ssh**/**key_bkup_ed25519.pub**, must be copied to the account 'bkupacct' on the remote system and placed in **~**/**.ssh**/**authorized_keys** in the correct place. Then it is necessary that the following directories on the server are owned by root and belong to the group 'backups' and are group readable, but not group writable, and definitely not world readable: **~** and **~**/**.ssh**/. Same for the file **~**/**.ssh**/**authorized_keys** there. (This also assumes you are not also using ACLs) However this is only one way of many to set permissions on the remote system:

```
$ sudo chown root:bkupacct ~
$ sudo chown root:bkupacct ~/.ssh/
```

17 http://linux.die.net/man/1/rsync
18 http://www.openbsd.org/cgi-bin/man.cgi/OpenBSD-current/man1/ssh.1
19 http://linux.die.net/man/1/rsync
20 http://linux.die.net/man/1/rsync
21 http://man.openbsd.org/ssh.1
22 https://en.wikibooks.org/wiki/OpenSSH/Cookbook/Public_Key_Authentication

```
$ sudo chown root:bkupacct ~/.ssh/authorized_keys
$ sudo chmod u=rwx,g=rx,o= ~
$ sudo chmod u=rwx,g=rx,o= ~/.ssh/
$ sudo chmod u=rwx,g=r,o=  ~/.ssh/authorized_keys
```

Now the configuration can begin.

Step 1: Configure sudoers(5)[23] so that rsync(1)[24] can work with sudo(8)[25] on the remote host. In this case data is staying on the remote machine. The group 'backups' will temporarily need full access in order to find and set specific options used later in locking this down.

```
%backups ALL=(root:root) NOPASSWD: /usr/bin/rsync
```

That is a transitory step and it is important that line should not be left in place as-is for any length of time.

However, while it is in place, ensure that rsync(1)[26] works with sudo(8)[27] by testing it with the –**rsync-path** option.

```
$ rsync --rsync-path='sudo rsync' \
-aHv bkupacct@www.example.org:/source/directotry/ /destination/directory/
```

The transfer should run without errors, warnings, or extra password entry.

Step 2: Next, do the same transfer again but using the key for authentication to make sure that the two can be used together.

```
$ rsync -e 'ssh -i ~/.ssh/key_bkup_ed25519' --rsync-path='sudo rsync' \
-aHv bkupacct@www.example.org:/source/directory/ /destination/directory/
```

Again, see the section on Public Key Authentication[28] if necessary.

Step 3: Now collect the connection details. They are needed to tune sudoers(5)[29] appropriately.

```
$ rsync -e 'ssh -E ssh.log -v -i ~/.ssh/key_bkup_ed25519' \
--rsync-path='sudo rsync' \
-aHv bkupacct@www.example.org:/source/directory/ /destination/directory/

$ grep -i 'sending command' ssh.log
```

The second command, the one with grep(1)[30], ought to produce something like the following:

23 http://linux.die.net/man/5/sudoers
24 http://linux.die.net/man/1/rsync
25 http://linux.die.net/man/8/sudo
26 http://linux.die.net/man/1/rsync
27 http://linux.die.net/man/8/sudo
28 https://en.wikibooks.org/wiki/OpenSSH/Cookbook/Public_Key_Authentication
29 http://linux.die.net/man/5/sudoers
30 http://man.openbsd.org/grep.1

```
debug1: Sending command: rsync --server --sender -vlHogDtpre.iLsfxCIvu .
 /source/directory/
```

The long string of letters and the directory are important to note because those will be used to tune sudoers(5)[31] a little. Remember that in these examples, the data gets copied from /**source**/**directory**/ on the remote machine to /**destination**/**directory**/ locally.

Here are the settings which match the formula above, assuming the account is in the group *backups*:

```
%backups ALL=(root:root) NOPASSWD: /usr/bin/rsync --server --sender
 -vlHogDtpre.iLsfxCIvu . /source/directory/
```

That line adjusts sudoers(5)[32] so that the backup account has enough access to run rsync(1)[33] as root but only in the directories it is supposed to run in and without free-rein on the system.

More refinements may come later, but those are the basics for locking sudoers(5)[34] down. At this point you are almost done, although the process can be automated much further. Be sure that the backed up data is not accessible to others once stored locally.

Step 4: Test rsync(1)[35] with sudo(8)[36] over ssh(1)[37] to verify that the settings made in sudoers(5)[38] are correct.

```
$ rsync -e 'ssh -i ~/.ssh/key_bkup_ed25519' --rsync-path='sudo rsync' \
-aHv bkupacct@www.example.org:/source/directory/ /destination/directory/
```

The backup should run correctly at this point.

Step 5: Finally it is possible to lock that key into just the one task by prepending restrictions using the **command="..."** option in the **authorized_keys** file. The explanation for that is found in sshd(8)[39].

```
command="/usr/bin/rsync --server --sender -vlHogDtpre.iLsfxCIvu .
 ./source/directory" ssh-ed25519 AAAAC3Nz...aWi
```

Thereafter that one key functions only for the backup. It's an extra layer upon the settings already made in the sudoers(5)[40] file.

31 http://linux.die.net/man/5/sudoers
32 http://linux.die.net/man/5/sudoers
33 http://linux.die.net/man/1/rsync
34 http://linux.die.net/man/5/sudoers
35 http://linux.die.net/man/1/rsync
36 http://linux.die.net/man/8/sudo
37 http://man.openbsd.org/ssh.1
38 http://linux.die.net/man/5/sudoers
39 http://man.openbsd.org/sshd.8
40 http://linux.die.net/man/5/sudoers

Thus you are able to do automated remote backup using rsync(1)[41] with root level access yet avoiding remote root login. Nevertheless keep close tabs on the private key since it can still be used to fetch the remote backup and that may contain sensitive information anyway.

From start to finish, the process requires a lot of attention to detail, but is quite doable if taken one step at a time. Setting up backups going the reverse direction is quite similar. When going from local to remote the —**sender** option will be omitted and the directories will be different.

16.1.3 Other Implementations of the Rsync Protocol

openrsync(1)[42] is a clean room reimplementation[43] of version 27 of the Rsync protocol as supported by the samba.org implementation of rsync(1)[43]. It has been in OpenBSD's base system since OpenBSD version 6.5. It is invoked with a different name, so if it is on a remote system and samba.org's rsync(1)[44] is on the local system, the –**rsync-path** option must be point to it by name:

```
$ rsync -a -v -e 'ssh -i key_rsa' \
    --rsync-path=/usr/bin/openrsync \
    fred@server.example.org:/var/www/ \
    /home/fred/www/
```

Going the other direction, starting with openrsync(1)[45] and connecting to rsync(1)[46] on the remote system, needs no such tweaking.

16.2 Backup Using tar(1)

A frequent choice for creating archives is tar(1)[47]. But since it copies whole files and directories, rsync(1)[48] is usually much more efficient for updates or incremental backups.

The following will make a tarball of the directory /**var**/**www**/ and send it via stdout on the local machine into **sdtin** on the remote machine via a pipe into ssh(1)[49] where, it is then directed into the file called **backup.tar**. Here tar(1)[50] runs on a local machine and stores the tarball remotely:

```
$ tar cf - /var/www/ | ssh -l fred server.example.org 'cat > backup.tar'
```

There are almost limitless variations on that recipe:

41 http://linux.die.net/man/1/rsync
42 http://man.openbsd.org/openrsync.1
43 http://linux.die.net/man/1/rsync
44 http://linux.die.net/man/1/rsync
45 http://man.openbsd.org/openrsync.1
46 http://linux.die.net/man/1/rsync
47 http://man.openbsd.org/tar.1
48 http://linux.die.net/man/1/rsync
49 http://man.openbsd.org/ssh.1
50 http://man.openbsd.org/tar.1

```
$ tar zcf - /var/www/ /home/*/www/ \
    |  ssh -l fred server.example.org 'cat > $(date +"%Y-%m-%d").tar.gz'
```

That example does the same, but also gets user WWW directories, compress the tarball using gzip(1)[51], and label the resulting file according to the current date. It can be done with keys, too:

```
$ tar zcf - /var/www/ /home/*/www/ \
    |  ssh -i key_rsa -l fred server.example.org 'cat > $(date +"%Y-%m-%d").tgz'
```

And going the other direction is just as easy for tar(1)[52] to find what is on a remote machine and store the tarball locally.

```
$ ssh fred@server.example.org 'tar zcf - /var/www/' >  backup.tgz
```

Or here is a fancier example of running tar(1)[53] on the remote machine but storing the tarball locally.

```
$ ssh -i key_rsa -l fred server.example.org 'tar jcf - /var/www/ /home/*/www/' \
    > $(date +"%Y-%m-%d").tar.bz2
```

So in summary, the secret to using tar(1)[54] for backup is the use of **stdout** and **stdin** to effect the transfer through pipes and redirects.

16.2.1 Backup of Files With tar(1) But Without Making A Tarball

Sometimes it is necessary to just transfer the files and directories without making a tarball at the destination. In addition to writing to **stdin** on the source machine, tar(1)[55] can read from **stdin** on the destination machine to transfer whole directory hierarchies at once.

```
$ tar zcf - /var/www/ | ssh -l fred server.example.org "cd /some/path/; tar zxf
 -"
```

Or going the opposite direction, it would be the following.

```
$ ssh 'tar zcf - /var/www/' | (cd /some/path/; tar zxf - )
```

However, these still copy everything each time they are run. So rsync(1)[56] described above in the previous section might be a better choice in many situations, since on subsequent runs it only copies the changes. Also, depending on the type of data network conditions, and CPUs available, compression might be a good idea either with tar(1)[57] or ssh(1)[58] itself.

51 http://man.openbsd.org/gzip.1
52 http://man.openbsd.org/tar.1
53 http://man.openbsd.org/tar.1
54 http://man.openbsd.org/tar.1
55 http://man.openbsd.org/tar.1
56 http://linux.die.net/man/1/rsync
57 http://man.openbsd.org/tar.1
58 http://man.openbsd.org/ssh.1

16.3 Backup Using dump

Using dump(8)[59] remotely is like using tar(1)[60]. One can copy from the remote server to the local server.

```
$ ssh -t source.example.org 'sudo dump -0an -f - /var/www/ | gzip -c9' >
 backup.dump.gz
```

Note that the password prompt for sudo(8)[61] might not be visible and it must be typed blindly.

Or one can go the other direction, copying from the locate server to the remote:

```
$ sudo dump -0an -f - /var/www/ | gzip -c9 | ssh target.example.org 'cat >
 backup.dump.gz'
```

Note again that the password prompt might get hidden in the initial output from dump(8)[62]. However, it's still there, even if not visible.

16.4 Backup Using zfs(8) Snapshots

OpenZFS[63] can easily make either full or incremental snapshots as a beneficial side effect of copy-on-write. These snapshots can be sent over SSH to or from another system. This method works equally well for backing up or restoring data. However, bandwidth is a consideration and the snapshots must be small enough to be feasible for the actual network connection in question. OpenZFS supports compressed replication such that the blocks which have been compressed on the disk remain compressed during transfer, reducing the need to recompress using another process. The transfers can be to or from either a regular file or another OpenZFS file system. It should be obvious but it is important to remember that smaller snapshots use less bandwidth and thus transfer more quickly than larger ones.

A full snapshot is required first because incremental snapshots only contain a partial set of data and require that the foundation upon which they were formed exists. The following uses zfs(8)[64] to make a snapshot named *20210326* of a dataset named *site01* in a pool named *web*.

```
$ zfs snapshot -r web/site01@20210326
```

The program itself will most likely be in the */sbin/* directory and either the PATH environment variable needs to include it or else the absolute path should be used instead. Incremental snapshots can subsequently be built upon the initial full snapshot by using the **-i** option. However, the ins and outs of OpenZFS management are far outside the scope of this book. Just the two methods for transfer between systems will be examined here. The

59 http://man.openbsd.org/dump.8
60 http://man.openbsd.org/tar.1
61 http://linux.die.net/man/8/sudo
62 http://man.openbsd.org/dump.8
63 https://openzfs.org/
64 https://linux.die.net/man/8/zfs

one method is using an intermediate file and the other is more direct using a pipe. Both use zfs send[65] and zfs receive[66] and the accounts involved must have the correct privileges in the OpenZFS Delegation System. For sending, it will be **send** and **snapshot** for the relevant OpenZFS pool. For receiving, it will be **create**, **mount**, and **receive** for the relevant pool.

16.4.1 OpenZFS To And From A Remote File System Via A File

A snapshot can be transferred to a file on a local or remote system over SSH. This method does not need privileged access on either system, but the account running zfs[67] must have the correct internal OpenZFS permissions as granted by zfs allow[68]. Here a very small snapshot is downloaded from the remote system to a local file:

```
$ ssh fred@server.example.org '/sbin/zfs send -v web/site01@20210326' >
 site01.openzfs
full send of web/site01@20210326 estimated size is 1.72M
total estimated size is 1.72M
```

If incremental snapshot are copied, the full snapshot on which they are based needs to be copied also. So care should be taken to ensure that this is a full snapshot and not just an incremental snapshot.

Later, restoring the snapshot is matter of going the reverse direction. In this case the data is retrieved from the file and sent to zfs(8)[69] over SSH.

```
$ cat site01.openzfs | ssh fred@server.example.org '/sbin/zfs receive -v -F
 web/site01@20210326'
receiving full stream of web/site01@20210326 into web/site01@20210326
received 1.85M stream in 6 seconds (316K/sec)
```

This is possible because the channel is 8-bit-clean when started without a PTY as happens when invoking programs directly at run time. Note that the targeted OpenZFS data set must be umounted using zfs(8)[70] first. Then after the transfer it must be mounted again.

The Other Direction

Transferring from the local system to the remote is a matter of changing around the order of the components.

```
$ /sbin/zfs send -v web/site01@20210326 | ssh fred@server.example.org 'cat >
 site01.openzfs'
full send of web/site01@20210326 estimated size is 1.72M
total estimated size is 1.72M
```

Then similar changes are needed to restore from the remote to the local.

65 https://linux.die.net/man/8/zfs
66 https://linux.die.net/man/8/zfs
67 https://linux.die.net/man/8/zfs
68 https://linux.die.net/man/8/zfs
69 https://linux.die.net/man/8/zfs
70 https://linux.die.net/man/8/zfs

```
$ ssh fred@server.example.org 'cat site01.openzfs' | /sbin/zfs receive -v -F
 web/site01@20210326'
receiving full stream of web/site01@20210326 into web/site01@20210326
received 1.85M stream in 6 seconds (316K/sec)
```

As usual, to avoid using the root account for these activities, the account running zfs(8)[71] must have the right levels of access within the OpenZFS Delegation System.

16.4.2 OpenZFS Directly To And From A Remote File System

Alternatively that snapshot can be transferred over SSH to a file system on the remote computer. This method needs privileged access and will irrevocably replace any changes made on the remote system since the snapshot.

```
$ zfs send -v pool/www@20210322 | ssh fred@server.example.org 'zfs receive -F
 pool/www@20210322'
```

So if removable hard drives are used on the remote system, this can update them.

```
$ ssh fred@server.example.org 'zfs send -v pool/www@20210322' | zfs receive -F
 pool/www@20210322
```

Again, the remote account must already have been permitted the necessary internal ZFS permissions.

The Other Direction

Again, to go the other direction, from a remote system to a local one, it is a matter of changing around the order of the components.

```
$ ssh fred@server.example.org 'zfs send -v pool/www@20210322' | zfs receive -F
 pool/www@20210322
```

And,

```
$ zfs send -v pool/www@20210322 | ssh fred@server.example.org 'zfs receive -F
 pool/www@20210322'
```

Again, working with the OpenZFS Delegation System can avoid the need for root access on either end of the transfer.

16.4.3 Buffering OpenZFS Transfers

Sometimes the CPU and network will alternate being the bottleneck during the file transfers. The **mbuffer(1)** utility can allow a steady flow of data [44] even when the CPU gets ahead of the network. The point is to leave a big enough buffer for there to always be some data transferring over the net even while the CPU is catching up.

71 https://linux.die.net/man/8/zfs

```
$ cat site01.zfs | mbuffer -s 128k -m 1G \
| ssh fred@server.example.org 'mbuffer -s 128k -m 1G | /sbin/zfs receive -v -F
 web/site01'

summary: 1896 kiByte in  0.2sec - average of 7959 kiB/s
receiving full stream of web/site01@20210326 into web/site01@20210326
in @ 2556 kiB/s, out @ 1460 kiB/s, 1024 kiB total, buffer   0% full
summary: 1896 kiByte in  0.8sec - average of 2514 kiB/s
received 1.85M stream in 2 seconds (948K/sec)
```

Further details of working with OpenZFS and managing its snapshots are outside the scope of this book. Indeed. there are whole guides, tutorials, and even books written about OpenZFS.

17 File Transfer with SFTP

Basic SFTP service requires no additional setup, it is a built-in part of the OpenSSH server and it is the subsystem sftp-server(8)[1] which then implements an SFTP file transfer. See the manual page for sftp-server(8)[2]. Alternately, the subsystem internal-sftp[3] can implement an in-process SFTP server which may simplify configurations using **ChrootDirectory** to force a different filesystem root on clients.

On the client, the same options and tricks are available for SFTP as for the regular SSH clients. However, some client options may have to be specified with the full option name using the **-o** argument. For many dedicated graphical SFTP clients, it is possible to use a regular URL to point to the target. Many file managers nowadays have built-in support for SFTP. See the section "GUI Clients[4]" above.

17.1 Basic SFTP

SFTP provides a very easy to use and very easy to configure option for accessing a remote system. Just to say it again, regular SFTP access requires no additional changes from the default configuration. The usual clients can be used or special ones like sshfs(1)[5].

17.1.1 Automated SFTP

SFTP uploads or downloads can be automated. The prerequisite is key-based authentication[6]. Once key-based authentication is working, a batch file can be used to carry out activities via SFTP. See the *batchfile* option **-b** in sftp(1)[7] for details.

```
$ sftp -b /home/fred/cmds.batch -i /home/fred/.ssh/foo_key_rsa
 server.example.org:/home/fred/logs/
```

If a dash (**-**) is used as the batch file name, SFTP commands will be read from **stdin**.

```
$ echo "put /var/log/foobar.log" | sftp -b - -i /home/fred/.ssh/foo_key_rsa
 server.example.org:/home/fred/logs/
```

1 http://man.openbsd.org/sftp-server.8
2 http://man.openbsd.org/sftp-server.8
3 https://en.wikibooks.org/wiki/OpenSSH/Server#The_SFTP_Server_Subsystem
4 https://en.wikibooks.org/wiki/OpenSSH/Client_Applications#GUI_Clients
5 http://linux.die.net/man/1/sshfs
6 https://en.wikibooks.org/wiki/OpenSSH/Cookbook/Public_Key_Authentication
7 http://man.openbsd.org/sftp.1

More than one SFTP command can be sent, but then it is better to use the batch file mode.

```
$ echo -e "put /var/log/foobar.log\nput /var/log/munged.log" | sftp -b - -i
 /home/fred/.ssh/foo_key_rsa server.example.org:/home/fred/logs/
```

The batch file mode can be very useful in cron jobs and in scripting.

17.2 SFTP-only Accounts

Using the **Match** directive in sshd_config(5)[8], it is possible to limit members of a specific group to using only SFTP for interaction with the server.

```
Subsystem sftp internal-sftp

Match Group sftp-only
    AllowTCPForwarding no
    X11Forwarding no
    ForceCommand internal-sftp
```

Note that disabling TCP forwarding does not improve security unless users are also denied shell access, as they can in principle install their own forwarders.

See PATTERNS in ssh_config(5)[9] for more information on patterns available to **Match**.

It is common for a group of accounts to need to read and write files to their home directories on the server while having little or no reason to access the rest of the file system. SFTP provides a very easy to use and very easy to configure chroot. In some cases, it is enough to chroot users to their home directories. This may not be as straight forward because in most cases home directories aren't owned by root and allow writing by at least one user. However, since SFTP chroot requires that the chroot target directory and all parent directories are owned by root and not writable by any others, this causes some difficulty but is necessary. Without ownership restrictions, it is quite feasible to escape the chroot. [45]

One way around the difficulties imposed by this restriction is to have the home directory owned by root and have it populated with a number of other directories and files that are owned by the regular account to which the user can actually write to.

```
Match Group sftp-only
    ChrootDirectory %h
    AllowTCPForwarding no
    X11Forwarding no
    ForceCommand internal-sftp
```

In that case the root user will have to populate the target directory with the needed files and subdirectories and then change their ownership to that of the unprivileged account.

8 http://man.openbsd.org/sshd_config.5
9 http://man.openbsd.org/ssh_config.5

17.2.1 Three Ways of Setting Permissions for Chrooted SFTP-Only Accounts

There are at least three other ways set permissions for home directories for chrooted SFTP-only accounts. They each have strong advantages and a few drawbacks and thus will suit some particular situations but not others.

Option 1: Automatically Setting the Start Directory

If it is not practical to have the various home directories owned by root, a compromise can be made. **ChrootDirectory** can point to /**home**, which must be owned by root anyway, and then **ForceCommand** can then designate the user's home directory as the starting directory using the **-d** option.

```
Match Group sftp-only
        ChrootDirectory /home/
        ForceCommand internal-sftp -d %u
```

Nesting another directory under the chroot is another way to do this. The subdirectory would be writable by the unprivileged account, the chroot target would not.

```
Match Group sftp-only
        ChrootDirectory /home/%u
        AllowTCPForwarding no
        X11Forwarding no
        ForceCommand internal-sftp -d %u
```

If it is necessary to hide the contents of the home directories from other users, chmod(1)[10] can be used. Permissions could be 0111 for /**home** and 0750, 0700, 0770, or 2770, and so on for the home directories, be sure to check the group memberships as well.

Option 2: Nested Home Directories

Alternately, for a similar effect but with more isolation, home directories can be nested one level deeper for the chrooted accounts. Note the ownership and permissions for the following directories:

```
$ ls -lhd /home/ /home/*/ /home/*/*/
drwxr-xr-x  4 root  root  4.0K Aug  4 20:47 /home/
drwxr-x---  3 root  user1 4.0K Aug  4 20:47 /home/user1/
drwxr-x---  3 root  user2 4.0K Aug  4 20:47 /home/user2/
drwxr-x--- 14 user1 user1 4.0K Aug  4 20:47 /home/user1/user1/
drwxr-x--- 14 user2 user2 4.0K Aug  4 20:47 /home/user2/user2/
```

Then the **ChrootDirectory** directive can lock the user to the directory above their home and the **ForceCommand** directive can put the user in their own home directory using the **-d** option. Once logged in they can only ever see their own files. This arrangement also makes it easier to add chrooted shell access later as system directories can be added to the chroot without being available to other accounts.

10 http://man.openbsd.org/chmod.1

Another common case is to chroot access to a web server's document root or server root. If each site has its own hierarchy under /**var**/**www**/ such as /**var**/**www**/**site1**/ then chroot can be put to use as follows:

```
Match Group team1
    ChrootDirectory /var/www/
    ForceCommand internal-sftp -d site1

Match Group team2
    ChrootDirectory /var/www/
    ForceCommand internal-sftp -d site2
```

The site directories can then be group writable while the parent /**var**/**www**/ remains read-only for non-root users.

Option 3: Split Ownership of the Home Directory and Its Contents

As seen above there are several ways to deal with a chrooted SFTP service. A third way would be to set the directory to be owned by root, in another group, but not writable by that group. So with the following settings, the account 'fred' can log in and use any of the pre-made subdirectories or files in the usual way, but cannot add anything to the chroot target itself.

```
$ ls -lhd /home/ /home/fred/ /home/fred/*
drwxr-xr-x  68 root root 4.0K Sep 4 15:40 /home/
drwxr-xr-x  21 root fred 4.0K Sep 4 15:41 /home/fred/
drwxr-xr-x   8 fred fred 4.0K Sep 4 15:44 /home/fred/Documents
drwxr-xr-x   9 fred fred 4.0K Sep 4 15:41 /home/fred/Music
drwxr-xr-x 145 fred fred 4.0K Sep 4 15:41 /home/fred/Pictures
drwxr-xr-x   5 fred fred 4.0K Sep 4 15:41 /home/fred/Videos
drwxr-xr-x  98 fred fred 4.0K Sep 4 15:41 /home/fred/WWW
```

The corresponding lines for ssh_config(5)[11] would be the following, with the account 'fred' being a member of the 'team1' group:

```
Match Group team1
    ChrootDirectory /home/%u
    ForceCommand internal-sftp
```

This method is quick to set up but the drawback is that it requires system administrator intervention any time a new directory or file is to be added to the home directory. That requires root permission to do even if the new file or directory is changed to being owned by the unprivileged account.

17.2.2 Umask

Starting with OpenSSH 5.4, sftp-server(8)[12] can set a umask to override the default one set by the user's account. The in-process SFTP server, internal-sftp, accepts the same options as the external SFTP subsystem.

11 http://man.openbsd.org/ssh_config.5
12 http://man.openbsd.org/sftp-server.8

```
Subsystem sftp internal-sftp -u 0022
```

However it is important to remember that umasks only restrict permissions, never loosen them.

Earlier versions can do the same thing through the use of a helper script, but this complicates chrooted directories very much. The helper script can be a regular script or it can be embedded inline in the configuration file though neither works easily in a chroot jail. It's often easier to get a newer version of sshd(8)[13] which supports umask as part of the server's configuration. Here is an inline helper script for umask in OpenSSH 5.3 and earlier, based on one by gilles@

```
Subsystem sftp /bin/sh -c 'umask 0022; /usr/libexec/openssh/sftp-server'
```

Either way, this umask is server-side only. The original file permissions on the client side will usually, but not always, be used when calculating the final file permissions on the server. This depends on the client itself. Most clients pass the file permissions on to the server, FileZilla being a notable exception. As such, permissions can generally be tightened but not loosened. For example, a file that is mode 600 on the client will not be automatically become 664 or anything else less than the original 600 regardless of the server-side umask. That is unless the client does not forward the permissions, in which case only the server's umask will be used. So for most clients, if you want looser permissions on the uploaded file, change them on the client side before uploading.

17.2.3 Further Restricting Chrooted SFTP

There are several additional common scenarios where chrooted access is restricted further.

Chrooted SFTP to Shared Directories

Another common case is to chroot a group of users to different levels of the web server they are responsible for. For obvious reasons, symbolic links going from inside the jail to parts of the filesystem outside the chroot jail are not accessible to the chrooted users. So directory hierarchies must be planned more carefully if there are special combinations of access. See the earlier section on chrooted SFTP-only accounts.

In these kinds of directories, it may be useful to give different levels of access to more than just one group. In that case, ACLs might be needed.

Chrooted SFTP Accounts Accessible Only from Particular Addresses

More complex matching can be done. It is possible to allow a group of users to use SFTP, but not a shell login, only if they log in from a specific address or range of addresses. If they log in from the right addresses, then get SFTP and only SFTP, but if they try to log in from other addresses they will be denied access completely. Both conditions, the affirmative and negative matches, need to be accounted for.

13 http://man.openbsd.org/sshd.8

```
Subsystem sftp internal-sftp

Match Group sftp-only, Address 192.0.2.10
        AllowTCPForwarding no
        X11Forwarding no
        ForceCommand internal-sftp
        ChrootDirectory  /home/servers/

Match Group sftp-only, Address *,!192.0.2.10
        DenyGroups sftp-only
```

Note that for negation a wildcard must be specified first and then the address or range to be excluded following it. Mind the spaces or lack thereof. Similar matching can be done for a range of addresses by specifying the addresses in CIDR address/mask format, such as 192.0.2.0/24. Any number of criteria can be specified and only if all of them are met then the directive in the subsequent lines take effect.

The first **Match** block that fits is the one that takes effect, so care must be taken when constructing conditional blocks to make them fit the precise situation desired. Also, any situations that don't fit a **Match** conditional block will fall through the cracks. Those will get the general configuration settings whatever they may be. Specific user and source address combinations can be tested with the configurations using the **-T** and **-C** options with the server for more options. See the section Debugging a Server Configuration[14] for more.

Chrooted SFTP with Logging

If the **internal-sftp** in-process SFTP server is not used then the logging daemon must establish a socket in the chroot directory for the sftp-server(8)[15] subsystem to access as /**dev**/**log** See the section on Logging[16].

17.2.4 Chrooted Login Shells

Making a chroot jail for interactive shells is difficult. The chroot and all its components must be root-owned directories that are not writable by any other user or group. The **ChrootDirectory** must contain the necessary files and directories to support the user's session. For an interactive session this requires at least a shell, typically bash(1)[17], ksh(1)[18], or sh(1)[19], and basic device nodes inside /**dev** such as null(4)[20], zero(4)[21], stdin(4)[22], std-

14 https://en.wikibooks.org/wiki/OpenSSH/Logging#Debugging_a_server_configuration
15 http://man.openbsd.org/sftp-server.8
16 https://en.wikibooks.org/wiki/OpenSSH/Logging
17 http://linux.die.net/man/1/bash
18 http://man.openbsd.org/ksh.1
19 http://man.openbsd.org/sh.1
20 http://man.openbsd.org/null.4
21 http://man.openbsd.org/zero.4
22 http://man.openbsd.org/stdin.4

out(4)[23], stderr(4)[24], arandom(4)[25], and tty(4)[26] devices. Paths may contain the following tokens that are expanded at runtime once the connecting user has been authenticated: **%%** is replaced by a literal '**%**', **%h** is replaced by the home directory of the user being authenticated, and **%u** is replaced by the username of that user.

17.3 sshfs(1) - SFTP File Transfer Via Local Folders

Another way to transfer files back and forth, or even use them remotely, is to use sshfs(1)[27] It is a user-space file system client based on SFTP and utilizes the server's SFTP-subsystem. It can make a directory on the remote server accessible as if it were a directory on the local file system which can be accessed by any program. The user must have read-write privileges for mount point to use sshfs(1)[28].

The following creates the mount point, **mountpoint**, in the home directory if none exists. Then sshfs(1)[29] mounts the remote server.

```
$ test -d ~/mountpoint || mkdir --mode 700 ~/mountpoint
$ sshfs fred@server.example.org:. ~/mountpoint
```

Reading or writing files to the mount point is actually transferring data to or from the remote system. The amount of bandwidth consumed by the transfers can be reduced using compression. That can be important if the network connection has bandwidth caps or per-unit fees. However, if speed is the only issue, compression can make the transfer slower if the processors on either end are busy or not powerful enough. About the only way to be sure is to test and see which method is faster. Below, compression is specified with **-C**.

```
$ sshfs -C fred@server.example.org:. ~/mountpoint
```

Or try with debugging output:

```
$ sshfs -o sshfs_debug fred@server.example.org:. /home/fred/mountpoint
```

Named pipes will not work over sshfs(1)[30]. Use `fusermount -u` to umount these remote directories and close the SFTP session.

23 http://man.openbsd.org/stdout.4
24 http://man.openbsd.org/stderr.4
25 http://man.openbsd.org/arandom.4
26 http://man.openbsd.org/tty.4
27 http://linux.die.net/man/1/sshfs
28 http://linux.die.net/man/1/sshfs
29 http://linux.die.net/man/1/sshfs
30 http://linux.die.net/man/1/sshfs

17.3.1 Using sshfs(1) With A Key

The **ssh_command** option is used to pass parameters on to ssh(1)[31]. In this example it is used to have ssh(1)[32] point to a key used for authentication to mount a remote directory, /**usr**/**src**, locally as /**home**/**fred**/**src**.

```
$ sshfs -o ssh_command="ssh -i /home/fred/.ssh/id_rsa"
 fred@server.example.org:/usr/src /home/fred/src/
```

If a usable key is already loaded into the agent, then ssh(1)[33] should find it and use it on behalf of sshfs(1)[34] without needing intervention.

31 http://man.openbsd.org/ssh.1
32 http://man.openbsd.org/ssh.1
33 http://man.openbsd.org/ssh.1
34 http://linux.die.net/man/1/sshfs

18 Public Key Authentication

Authentication keys can improve efficiency, if done properly. As a bonus advantage, the passphrase and private key never leave the client[46]. Key-based authentication is generally recommended for outward facing systems so that password authentication can be turned off.

18.1 Key-based authentication

OpenSSH can use public key cryptography for authentication. In public key cryptography, encryption and decryption are asymmetric. The keys are used in pairs, a public key to encrypt and a private key to decrypt. The ssh-keygen(1)[1] utility can make RSA, Ed25519, ECDSA, Ed25519-SK, or ECDSA-SK keys for authenticating. Even though DSA keys can still be made, being exactly 1024 bits in size, they are no longer recommended and should be avoided. RSA keys are allowed to vary from 1024 bits on up. The default is now 3072. However, there is only limited benefit after 2048 bits and that makes elliptic curve algorithms preferable. ECDSA can be 256, 384 or 521 bits in size. Ed25519, Ed25519-SK, and ECDSA-SK keys each have a fixed length of 256 bits. Shorter keys are faster, but less secure. Longer keys are much slower to work with but provide better protection, up to a point.

Keys can be named to help remember what they are for. Because the key files can be named anything it is possible to have many keys each named for different services or tasks. The comment field at the end of the public key can also be useful in helping to keep the keys sorted, if you have many of them or use them infrequently.

The process of key-based authentication uses these keys to make a couple of exchanges using the keys to encrypt and decrypt some short message. At the start, a copy of the client's public key is stored on the server and the client's private key is on the client, both stay where they are. The private key never leaves the client. As the client first contacts the server, the server responds by using the client's public key to encrypt a random number and return that encrypted random number as a challenge to the client. The client responds to the challenge by using the matching private key to decrypt the message and extract the random number. The client then makes an MD5 hash of the session ID along with the random number from the challenge and returns that hash to the server. The server then makes its own hash of the session ID and the random number and compares that to the

1 `http://man.openbsd.org/ssh-keygen.1`

hash returned by the client. If there is a match, the login is allowed. If there is not a match, then the next of any public keys on the server registered as belonging to the same account is tried until either a match is found or all the keys have been tried or the maximum number of failures has been reached. [47]

When an agent is used on the client side to manage authentication, the process is similar. The difference is that ssh(1)[2] passes the challenge off to the agent which then calculates the response and passes it back to ssh(1)[3] which then passes the agent's response back to the server.

18.1.1 Basics of Public Key Authentication

A matching pair of keys is needed for public key authentication and ssh-keygen(1)[4] is used to make the key pair. Out of that pair the public key must be properly stored on the remote host. There on the server public key is added to the designated **authorized_keys** file for that remote user account. The private key stays stored safely on the client. Once the keys have been prepared they can be used again and again.

There are four steps in preparation for key-based authentication:

1) Prepare the directories where the keys will stay. If either the **authorized_keys** file or **.ssh** directory do not exist on either the remote machine or the **.ssh** directory on the remote machine, create them and set the permissions correctly. On the client only a directory is needed, but it should not be writable by any account except its owner:

```
$ mkdir ~/.ssh/
$ chmod 0700 ~/.ssh/
```

On the remote machine, the **.ssh** directory is needed as is a special file to store the public keys, the default is **authorized_keys**.

```
$ mkdir ~/.ssh/
$ touch ~/.ssh/authorized_keys
$ chmod 0700 ~/.ssh/
$ chmod 0600 ~/.ssh/authorized_keys
```

2) Create a key pair. The example here creates a Ed25519 key pair in the directory **~/.ssh** . The option **-t** assigns the key type and the option **-f** assigns the key file a name. It is good to give keys files descriptive names, especially if larger numbers of keys are managed. Below, the public key will be named **mykey_ed25510.pub** and the private key will be called **mykey_ed25519**. Be sure to enter a sound passphrase to encrypt the private key using 128-bit AES[5].

```
$ ssh-keygen -t ed25519 -f ~/.ssh/mykey_ed25519
```

Ed25519, Ed25519-SK, and ECDSA-SK keys have a fixed length. For RSA and ECDSA keys, the **-b** option sets the number of bits used.

2 http://man.openbsd.org/ssh.1
3 http://man.openbsd.org/ssh.1
4 http://man.openbsd.org/ssh-keygen.1
5 https://en.wikibooks.org/w/index.php?title=AES&action=edit&redlink=1

Since 6.5 a new private key format is available using a bcrypt(3)[6] key derivative function (KDF) to better protect keys at rest. This new format is always used for Ed25519 keys, and sometime in the future will be the default for all keys. But for right now it may be requested when generating or saving existing keys of other types via the **-o** option in ssh-keygen(1)[7].

```
$ ssh-keygen -o -b 4096 -t rsa -f ~/.ssh/mykey_rsa
```

Details of the new format are found in the source code in the file **PROTOCOL.key**.

3) Get the keys to the right places. Transfer only the public key to remote machine.

```
$ ssh-copy-id -i ~/.ssh/mykey_ed25519 fred@remotehost.example.org
```

If ssh-copy-id(1)[8] is not available, any editor that does not wrap long lines can be used. For example, nano(1)[9] can be started with the **-w** option to prevent wrapping of long lines. Or another way to set that permanently is by editing nanorc(5)[10] However the **authorized_keys** file is edited to add the key, the key itself must be in the file whole and unbroken on a single line.

Then if they are not already on the client, transfer both the public and private keys there. It is usually best to keep both the public and private keys together in the directory **~/.ssh/** , though the public key is not needed on the client after this step and can be regenerated if it is ever needed again.

4) Test the keys

While still logged in, use the client start another SSH session in a new window and try authenticating to the remote machine from the client using the private key.

```
$ ssh -i ~/.ssh/mykey_ed25519 -l fred remotehost.example.org
```

The option **-i** tells ssh(1)[11] which private key to try. Close the original SSH session only after verifying that the key-based authentication works.

Once key-based authentication has been verified to be working, it is possible to make permanent shortcuts on the client using ssh_config(5)[12], explained further below. In particular, see the **IdentityFile**, **IdentitiesOnly**, and **AddKeysToAgent** configuration directives, to name three.

➥**Troubleshooting of Key-based Authentication**:

If the server refuses to accept the key and fails over to the next authentication method (eg: "Server refused our key"), then there are several possible mistakes to look for on the server side.

6 http://man.openbsd.org/bcrypt.3
7 http://man.openbsd.org/ssh-keygen.1
8 https://linux.die.net/man/1/ssh-copy-id
9 http://linux.die.net/man/1/nano
10 http://linux.die.net/man/5/nanorc
11 http://man.openbsd.org/ssh.1
12 http://man.openbsd.org/ssh_config.5

One of the most common errors is that the file and directory permissions are wrong. The authorized key file must be owned by the user in question and not be group writable. Nor may the key file's directory be group or world writable.

```
$ chmod u=rwx,g=rx,o= ~/.ssh
$ chmod u=rw,g=,o=  ~/.ssh/authorized_keys
```

Another mistake that can happen is if the key inside the **authorized_keys** file on the remote host is broken by line breaks or has other whitespace in the middle. That can be fixed by joining up the lines and removing the spaces or by recopying the key more carefully.

And, though it should go without saying, the halves of the key pair need to match. The public key on the server needs to match the private key held on the client. If the public key is lost, then a new one can be generated with the **-y** option, but not the other way around. If the private key is lost, then the public key should be erased as it is no longer of any use. If many keys are in use for an account, it might be a good idea to add comments to them. On the client, it can be a good idea to know which server the key is for, either through the file name itself or through the comment field. A comment can be added using the **-C** option.

```
$ ssh-keygen -t ed25519 -f ~/.ssh/mykey_ed25519 -C "web server mirror"
```

On the server, it can be important to annotate which client they key is from if there is more than one public key there in an account. There the comment can be added to the authorized key file on the server in the last column if a comment does not already exist. Again, the format of the authorized keys file is given in the manual page for sshd(8)[13] in the section "AUTHORIZED_KEYS FILE FORMAT". If the keys are not labeled they can be hard to match, which might or might not be what you want.

Associating Keys Permanently with a Server

A key can be specified at run time, but to save retyping the same paths again and again, the **Host** directive in ssh_config(5)[14] can apply specific settings to a target host. In this case, by changing **~/.ssh/config** it is possible to assign particular keys to be tried automatically whenever making a connection to that specific host. After adding the following lines to **~/.ssh/config**, all that's needed is to type `ssh` *`web1`* to connect with the key for that server.

```
Host web1
    Hostname 198.51.100.32
    IdentitiesOnly yes
    IdentityFile /home/fred/.ssh/web_key_ed25519
```

Below **~/.ssh/config** uses different keys for *server* versus *server.example.org*, regardless whether they resolve to the same machine. This is possible because the host name argument given to ssh(1)[15] is not converted to a canonicalized host name before matching.

13 http://man.openbsd.org/sshd.8
14 http://man.openbsd.org/ssh_config.5
15 http://man.openbsd.org/ssh.1

```
Host server
    IdentitiesOnly yes
    IdentityFile /home/fred/.ssh/key_a_rsa

Host server.example.org
    IdentitiesOnly yes
    IdentityFile /home/fred/.ssh/key_b_rsa
```

In this example the shorter name is tried first, but of course less ambiguous shortcuts can be made instead. The configuration file gets parsed on a first-match basis. So the most specific rules go at the beginning and the most general rules go at the end.

Encrypted Home Directories

When using encrypted home directories the keys must be stored in an unencrypted directory. That means somewhere outside the actual home directory which means sshd(8)[16] needs to be configured appropriately to find the keys in that special location.

Here is one method for solving the access problem. Each user is given a subdirectory under /**etc**/**ssh**/**keys**/ which they can then use for storing their **authorized_keys** file. This is set in the server's configuration file /**etc**/**ssh**/**sshd_config**

```
AuthorizedKeysFile      /etc/ssh/keys/%u/authorized_keys
```

Setting a special location for the keys opens up more possibilities as to how the keys can be managed and multiple key file locations can be specified if they are separated by whitespace. The user does not have to have write permissions for the **authorized_keys** file. Only read permission is needed to be able to log in. But if the user is allowed to add, remove, or change their keys, then they will need write access to the file to do that.

One symptom of having an encrypted home directory is that key-based authentication only works when you are already logged into the same account, but fails when trying to make the first connection and log in for the first time.

Sometimes it is also necessary to add a script or call a program from /**etc**/**ssh**/**sshrc** immediately after authentication to decrypt the home directory.

Passwordless Login

One way of allowing passwordless logins is to follow the steps above, but simply do not enter a passphrase when asked for one while creating the key. Note that using keys that lack a passphrase is very risky, so the key files should be very well protected and kept track of. That includes that they only be used as single-purpose keys as described below. Timely key rotation becomes especially important. In general, it is not a good idea to make a key without a passphrase. A better solution is to have a passphrase and work with an authentication agent in conjunction with a single-purpose key. Most desktop environments launch an SSH agent automatically these days. It will be visible in the **SSH_AUTH_SOCK**

16 http://man.openbsd.org/sshd.8

environment variable if it is. On accounts with an agent, ssh-add(1)[17] can load private keys into an available agent.

```
$ ssh-add ~/.ssh/mykey_ed25519
```

Thereafter, the client will automatically check the agent for the key when appropriate. If there are many keys in the agent, it will become necessary to set **IdentitiesOnly**. See the above section on using **~/.ssh/config** for that. See [OpenSSH/Cookbook/Public_Key_Authentication#Key-based_Authentication_Using_an_Agent Key-based Authentication Using an Agent] below.

Requiring Both Keys and a Password

While users should have strong passphrases for their keys, there is no way to enforce or verify that. Indeed, since neither the private key nor its the passphrase ever leave the client machine there is nothing that the server can do to have any influence over that. Instead, it is possible to require both a key and a password. Starting with OpenSSH 6.2, it is possible for the server to require multiple authentication methods for login using the **AuthenticationMethods** directive.

```
AuthenticationMethods publickey,password
```

This example from [18] sshd_config(5)] requires that users first authenticate using a key and it only queries for a password if the key succeeds. Thus with that configuration it is not possible to get to the system password prompt without first authenticating with a valid key. Changing the order of the arguments changes the order of the authentication methods.

Requiring Two or More Keys

Since OpenSSH 6.8, the server now remembers which public keys have been used for authentication and refuses to accept previously-used keys. This allows a set up requiring that users authenticate using two different public keys, maybe one in the file system and the other in a hardware token.

```
AuthenticationMethods publickey,publickey
```

The **AuthenticationMethods** directive, whether for keys or passwords, can also be set on the server under a **Match** directive to apply only to certain groups or situations.

Requiring Certain Key Types For Authentication

Also since OpenSSH 6.8, the **PubkeyAcceptedKeyTypes** directive, later changed to **PubkeyAcceptedAlgorithms**, can specify which key algorithms are accepted for authentication. Those not in the comma-separated pattern list are not allowed.

17 http://man.openbsd.org/ssh-add.1
18 http://man.openbsd.org/sshd_config.5

```
PubkeyAcceptedAlgorithms
 ssh-ed25519*,ssh-rsa*,ecdsa-sha2*,sk-ssh-ed25519*,sk-ecdsa-sha2*
```

Either the actual key types or a pattern can be in the list. Spaces are not allowed in the pattern list. The exact list of key types supported for authentication can be found by the **-Q** option using the client. The following two lines are equivalent.

```
$ ssh -Q key-sig | sort

$ ssh -Q PubkeyAcceptedAlgorithms | sort
```

For host-based authentication, it is the **HostbasedAcceptedAlgorithms** directive which determines the key types which are allowed for authentication.

18.1.2 Key-based Authentication Using an Agent

When an authentication agent, such as ssh-agent(1)[19], is going to be used, it should generally be started at the beginning of a session and used to launch the login session or X-session so that the environment variables pointing to the agent and its UNIX-domain socket are passed to each subsequent shell and process. Many desktop distros do this automatically upon login or startup.

Starting an agent entails setting a pair of environment variables:

- SSH_AGENT_PID : the process id of the agent
- SSH_AUTH_SOCK : the filename and full path to the UNIX-domain socket

The various SSH and SFTP clients find these variables automatically and use them to contact the agent and try when authentication is needed. However, it is mainly SSH_AUTH_SOCK which is ever used. If the shell or desktop session was launched using ssh-agent(1)[20], then these variables are already set and available. If they are not available, then it is necessary to either set the variables manually inside each shell or for each application in order to use the agent or else to point to the agent's socket using the directive **IdentityAgent** in the client's configuration file.

Once an agent is available, a relevant private key needs to be loaded before the agent can be used. Once in the agent the private key can then be used many times. Private keys are loaded into an agent with ssh-add(1)[21].

```
$ ssh-add /home/fred/.ssh/mykey_ed25519
Identity added: /home/fred/.ssh/mykey_ed25519 (/home/fred/.ssh/mykey_ed25519)
```

Keys stay in the agent for as long as it is running unless specified otherwise. A timeout can be set either with the **-t** option when starting the agent itself or when actually loading the key using ssh-add(1)[22]. In either case, the **-t** option will set a timeout interval, after which the key will be purged from the agent.

19 http://man.openbsd.org/ssh-agent.1
20 http://man.openbsd.org/ssh-agent.1
21 http://man.openbsd.org/ssh-add.1
22 http://man.openbsd.org/ssh-add.1

```
$ ssh-add -t 1h30m /home/fred/.ssh/mykey_ed25519
Identity added: /home/fred/.ssh/mykey_ed25519 (/home/fred/.ssh/mykey_ed25519)
Lifetime set to 5400 seconds
```

The option **-l** will list the fingerprints of all of the identities in the agent.

```
$ ssh-add -l
256 SHA256:77mfUupj364g1WQ+O8NM1ELj0G1QRx/pHtvzvDvDlOk mykey for task x
 (ED25519)
3072 SHA256:7unq90B/XjrRbucm/fqTOJu0I1vPygVkN9FgzsJdXbk myotherkey rsa for task
 y (RSA)
```

It is also possible to remove individual identities from the agent using **-d** which will remove them one at a time if identified by file name, but only if the file name is given and without the file name of the private key to be remove, **-d** will fail silently. Using **-D** instead will remove all of them at once without needing to specify any by name.

By default ssh-add(1)[23] uses the agent connected via the socket named in the environment variable **SSH_AUTH_SOCK**, if it is set. Currently, that is its only option. However, for ssh(1)[24] an alternative to using the environment variable is the client configuration directive **IdentityAgent** which tells the SSH clients which socket to use to communicate with the agent. If both the environment variable and the configuration directive are available at the same time, then the value in **IdentityAgent** takes precedence over what's in the environment variable. **IdentityAgent** can also be set to *none* to prevent the connection from trying to use any agent at all.

The client configuration directive **AddKeysToAgent** can also be useful in getting keys into an agent as needed. When set, it automatically loads a key into a running agent the first time the key is called for if it is not already loaded. Likewise the **IdentitiesOnly** directive can ensure that the relevant key is offered on the first try. Rather than typing these out whenever the client is run, they can be added to **~/.ssh/config** and thereby added automatically for designated host connections.

Agent Forwarding

Agent forwarding is one means of passing through one or more intermediate hosts. However, the **-J** option for **ProxyJump** would be a safer option. See Passing Through a Gateway or Two[25] in the section on jump hosts about that. With agent forwarding, intermediate machines forward challenges and responses back and forth between the client and the final destination. This comes with some risks but eliminates the need for using passwords or holding keys on any of these intermediate machines.

A main advantage of agent forwarding is that the private key itself is not needed on any remote machine, thus hindering unwanted file system access to it. [48] Another advantage is that the actual agent to which the user has authenticated does not go anywhere and is thus less susceptible to analysis.

23 http://man.openbsd.org/ssh-add.1

24 http://man.openbsd.org/ssh.1

25 https://en.wikibooks.org/wiki/OpenSSH/Cookbook/Proxies_and_Jump_Hosts#Jump_Hosts_--_Passing_Through_a_Gateway_or_Two

One risk with agents is that they can be re-used to tailgate in if the permissions allow it. Keys cannot be copied this way, but authentication is possible when there are incorrect permissions. Note that disabling agent forwarding does not improve security unless users are also denied shell access, as they can always install their own forwarders.

The risks of agent forwarding can be mitigated by confirming each use of a key by adding the **-c** option when adding the key to the agent. This requires the SSH_ASKPASS variable be set and available to the agent process, but will generate a prompt on the host running the agent upon each use of the key by a remote system. So if passing through one or more intermediate hosts, it is usually better to instead have the SSH client use stdio forwarding with **-W** or **-J**.

On the client side agent forwarding is disabled by default and so if it is to be used it must be enabled explicitly. Put the following line in ssh_config(5)[26] to enable agent forwarding for a particular server:

```
Host gateway.example.org
ForwardAgent yes
```

On the server side the default configuration files allow authentication agent forwarding, so to use it, nothing needs to be done there, just on the client side. However, again, it would be preferable to take a look at **ProxyJump** instead.

Old Style, Somewhat Safer SSH Agent Forwarding

The best way to pass through one or more intermediate hosts is to use the **ProxyJump** option instead of authentication agent forwarding and thereby not risk exposing any private keys. If authentication agent forwarding must be used, then it would be advisable in the interest of following the principle of least privilege to forward an agent containing the minimum necessary number of keys. There are several ways to solve that.

In version 8.8 and earlier a partial solution is to make a one-off, ephemeral agent to hold just the one key or keys needed for the task at hand. Another partial solution would be to set up a user-accessible service at the operating system level and then use ssh_config[27] for the rest.

Automatically launching an ephemeral agent unique to each session can be done by crafting either a special shell alias or function to launch a single-use agent. Either the function or the alias can be written to require confirmation for each requested signature. The following example is an alias is based on an updated blog post by Vincent Bernat[49] on SSH agent forwarding:

```
$ alias assh="ssh-agent ssh -o AddKeysToAgent=confirm -o ForwardAgent=yes"
```

Note the use of ssh-agent(1)[28]. When invoking that alias, the SSH client will be launched with a unique, ephemeral supporting key agent. The alias sets up a new agent, including setting the two environment variables, and then sets two client options while calling the

26 http://man.openbsd.org/ssh_config.5
27 http://man.openbsd.org/ssh_config.5
28 http://man.openbsd.org/ssh-agent.1

client. This arrangement still checks with ssh_config(5)[29] for other options and settings. When the SSH session is finished the agent which launched it ends and goes away, thus cleaning up after itself automatically.

Another way is to rely on the client's configuration file for some of the settings. Such methods rely mostly on ssh_config(5)[30] but still require an independent method to launch an ephemeral agent because the OpenSSH client is already running by the time it reads the configuration file and is thus not affected by any changes to environment variables caused by the configuration file and it is through the environment variables that contain information about the agent. However, when the path to the UNIX-domain socket used to communicate with the authentication agent is decided in advance then the **IdentityAgent** option can point to it once the one-off agent[50] is actually launched. The following uses a specific agent's pre-defined socket whenever connecting to either of two particular domains:

```
Host *.wikimedia.org *.wmflabs.org
        User fred
        IdentitiesOnly yes
        IdentityFile %d/.ssh/id_cloud_01
        IdentityAgent /run/user/%i/ssh-cloud-01.socket
        ForwardAgent yes
        AddKeysToAgent yes
```

The **%d** stands for the path to the home directory and the **%i** stands for the user id (UID) for the current account. In some cases the **%i** token might also come in handy when setting the **IdentityAgent** option inside the configuration file. Again, be careful when forwarding agents with which keys are in the forwarded agent. See the section "TOKENS" in ssh_config(5)[31] for more such abbreviations.

With those configuration settings, the authentication agent must already be up and running and point to the designated socket prior to starting the SSH client for that configuration to work. Additionally, it should place the socket in a directory which is inaccessible to any other accounts. ssh-agent(1)[32] must use the **-a** option to name the socket:

```
$ ssh-agent -a /run/user/${UID}/ssh-cloud-01.socket
```

That agent configuration can be launched manually or via a script or service manager. However, in the interests of privacy and security in general, agent forwarding is to be avoided. The configuration directive **ProxyJump** is the best alternative and, on older systems, host traversal using **ProxyCommand** with netcat[33] are preferable. Again, see the section on Proxies and Jump Hosts[34] for how those methods are used.

New Style SSH Agent Destination Constraints

From 8.9 onward, ssh-agent(1)[35] will allow the agent to limit which hosts they will be used

29 http://man.openbsd.org/ssh_config.5
30 http://man.openbsd.org/ssh_config.5
31 http://man.openbsd.org/ssh_config.5
32 http://man.openbsd.org/ssh-agent.1
33 http://man.openbsd.org/nc.1
34 https://en.wikibooks.org/wiki/OpenSSH/Cookbook/Proxies_and_Jump_Hosts
35 http://man.openbsd.org/ssh-agent.1

for authentication as specified by ssh-add(1)[36] using the **-h** option. These constraints have been added through two agent protocol extensions and a modification to the public key authentication protocol. This feature may evolve, but for now the result is such that keys for account authentication can be loaded into the agent in four ways:

- no limits on forwarding (not recommended)
- local use only, these will not get forwarded
- forwarding, but only to specific remote hosts
- forwarding to specific remote hosts via specified routes

The intent is that the restrictions fail safely so that they do not allow authentication when one or more hosts in the route lack the needed protocol features. The destinations and routes cannot be modified once the keys are loaded, but multiple routes to the same destination can be loaded and the routes can be any number of hops. If the routes need changing, then the key must be reloaded into the agent with the new route or routes.

The general default for the client is to keep keys in the agent for local use only. However, that can be enforced explicitly by adding the **-a** option when starting the client or else setting the **ForwardAgent** directive in ssh_config(5)[37] to 'no' in the relevant configuration block.

In order to load keys for unlimited forwarding, which is not the best idea, just add them using ssh-add(1)[38] as normal. Then use the **-A** option with the client or set the **ForwardAgent** directive in ssh_config(5)[39] to 'yes' in the relevant configuration block.

In order to limit keys for connection only to a specific remote host, or to load keys for connection to a specific remote host with forwarding via one or more intermediate hosts, use he **-h** option when loading keys into the agent. Here the one key may be used only to connect to the specific destination:

```
$ ssh-agent -h server.example.org server.key.ed25519
```

If an intermediate system is passed through, the best way is to use **ProxyJump** which is the **-J** option for the SSH Client. If agent forwarding must be allowed then the tightest way is to constrain which systems may use the keys, again using the **-h** option.

```
$ ssh-agent -h middle.example.org -h "middle.example.org>server.example.org"
 server.key.ed25519
```

Multiple steps can be included, even multiple routes. They just have to be enumerated explicitly, though patterns may still be used for the destination hosts as well as specific names. Each host in the chain must support these protocol extensions for the connection to complete.

Any keys designated for forwarding are unusable for authentication on any other hosts than those which have been explicitly identified for forwarding. These permitted hosts are identified by host key or host certificate from the **known_hosts** file or another file

36 http://man.openbsd.org/ssh-add.1
37 http://man.openbsd.org/ssh_config.5
38 http://man.openbsd.org/ssh-add.1
39 http://man.openbsd.org/ssh_config.5

designated by the **-H** option when loading the key with ssh-add(1)[40]. If **-H** is not used at the time the keys are loaded into the agent, then the default known hosts file(s) will be used: **~/.ssh/known_hosts**, **/etc/ssh/ssh_known_hosts**, **~/.ssh/known_hosts2**, and **/etc/ssh/ssh_known_hosts2**.

In the case of keys, the **known_hosts** list must be maintained conscientiously [51], perhaps with the help of the **UpdateHostkeys** and **CanonicalizeHostname** client configuration directives. Use of certificates requires the agent to only need to be aware of the Certificate Authority (CA).

Again, see Passing Through a Gateway or Two[41] in the section on jump hosts about a way to pass through one or more intermediate machines without needing to forward an SSH agent.

Checking the Agent for Specific Keys

The ssh_add(1)[42] utility's **-T** option can test whether a specific private key is available in the agent or not by looking up the matching public key. That can be useful in a shell script.

```
#!/bin/sh

key=/home/fred/.ssh/some.key.ed25519.pub

if ssh-add -T ${key}; then
        echo "Key ${key} Found"
else
        echo "Key ${key} missing"
fi
```

Or it could be done with an alternate syntax just as well either in a script or in an interactive shell sessions,

```
$ key=/home/fred/.ssh/some.key.ed25519.pub
$ ssh-add -T ${key} && echo "Key found" || echo "Key missing"
```

However, if the desired result would be to add key to the agent then the **AddKeysToAgent** client configuration option can ensure that a specific key is added to the SSH agent upon first use during any given login session. That can be done using **-o AddKeysToAgent=yes** as a run-time argument, or by modifying ssh_config(5)[43] as appropriate:

```
Host www
        HostName www.example.com
        IdentityFile %d/.ssh/www.ed25519
        IdentitiesOnly yes
        AddKeysToAgent yes
```

With those options in the configuration file, the first time `ssh www` is run the specified key will get added to the agent and remain available.

40 http://man.openbsd.org/ssh-add.1
41 https://en.wikibooks.org/wiki/OpenSSH/Cookbook/Proxies_and_Jump_Hosts#Jump_Hosts_--_Passing_Through_a_Gateway_or_Two
42 http://man.openbsd.org/ssh_add
43 http://man.openbsd.org/ssh_config

18.1.3 Key-based Authentication Using A Hardware Security Token

While stand-alone keys have been around for a long time, it has been possible since version 8.2 to use keys backed by hardware security tokens, such as OnlyKey, Yubikey, or many others, though the FIDO2 protocol. The Universal 2nd Factor (U2F) authentication is supported directly in OpenSSH through FIDO2 and does not need third party software. At the moment there are two types of hardware backed keys, ECDSA-SK and Ed25519-SK, but only the latest hardware tokens support the latter. If the key Ed25519-SK format is not supported by the token's firmware, then the following error message will be presented when attempts to use that key type are made:

```
Key enrollment failed: invalid format
```

If supported, either key type can be created with ssh-keygen(1)[44]. The steps are almost identical to creating normal keys but the token must be available to the system (plugged in) first. Then if called for, the token's PIN must be entered and the token touched or otherwise activated. After that, the key creation proceeds as normal. Mind the key type as specified by the **-t** option.

```
$ ssh-keygen -t ed25519-sk -f /home/fred/.ssh/server.ed25519-sk -C "web server
 for fred"
Generating public/private ed25519-sk key pair.
You may need to touch your authenticator to authorize key generation.
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/fred/.ssh/server.ed25519-sk
Your public key has been saved in /home/fred/.ssh/server.ed25519-sk.pub
The key fingerprint is:
SHA256:41wVVDnKJ9gKr2Sj4CFuYMhcNvYebZ6zq0PWyP4rRDo web server
The key's randomart image is:
+[ED25519-SK 256]-+
|           .o... |
|             .o  |
|           +.. . |
|   = .  . ..= .  |
|+ + * + So.. o   |
|o+.EoO *+oo      |
|.o oBo+++o       |
|  o .=.+.        |
| .   .===        |
+----[SHA256]-----+
```

Once created, the public and private key files get handled like with any other type of key. But when authenticating, the hardware token must be present and activated when called for.

```
$ ssh -i /home/fred/.ssh/server.ed25519-sk server.example.org
Enter passphrase for key '/home/fred/.ssh/server.ed25519-sk':
Confirm user presence for key ED25519-SK
 SHA256:41wVVDnKJ9gKr2Sj4CFuYMhcNvYebZ6zq0PWyP4rRDo
```

The resulting private key file is not actually the key itself but instead a ”key handle” which is used by the hardware security token to derive the real private key on demand at the time it is actually used[52]. As a result, the hardware-backed private key file is useless without

44 http://man.openbsd.org/ssh-keygen.1

the accompanying hardware token. This also means that these key files are not portable across hardware tokens, say when having multiple tokens in reserve or as backup, even when used by the same account. So when multiple hardware tokens are in use, different key pairs must be generated for each token.

Hardware Security Token Resident Private Key

It is possible to store the private key within the token itself, but for the moment it cannot be used directly from inside the token and must first be saved as a file. Also, the key can only be loaded into the FIDO authenticator at the time of creation using the **-O resident** option with ssh-keygen(1)[45]. Otherwise, the process is the same as above.

```
$ ssh-keygen -O resident -t ed25519-sk -f /home/fred/.ssh/server.ed25519-sk -C
 "web server for fred"
. . .
```

When needed, the resident key can be extracted from the FIDO2 hardware token and saved into a file using the **-K** option. At this stage a passphrase can be added to the file, but no passphrase is kept within the token itself, only an optional PIN protects the key there.

```
$ ssh-keygen -K
Enter PIN for authenticator:
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Saved ED25519-SK key to id_ed25519_sk_rk

$ mv -i id_ed25519_sk_rk /home/fred/.ssh/server.ed25519-sk
```

Since the output file name is fixed, any pre-existing file with that name can get overwritten but there will be a warning first. However, it is not recommended to keep the key on the hardware token because it provides more protection when kept separately.

18.2 Single-purpose Keys

Tailored single-purpose keys can eliminate use of remote root logins for many administrative activities. A finely tailored **sudoers** is needed along with an unprivileged account. When done right, it gives just enough access to get the job done, following the security principle of Least Privilege.

Single-purpose keys are accompanied by use of either the **ForceCommand** directive in sshd_config(5)[46] or the **command="..."** directive inside the **authorized_keys** file. The method is to generate a new key pair, transfer the public key to **authorized-keys** on the remote system, and then prepend the appropriate command or script there to the line with the key.

```
$ grep '^command' ~/.ssh/authorized_keys
command="/usr/local/bin/somescript.sh" ssh-ed25519
 AAAAC3NzaC1lZDI1NTE5AAAAIEcgdzDvSebOEjuegEx4W1I/aA7MM3owHfMr9yg2WH8H
```

45 http://man.openbsd.org/ssh-keygen.1
46 http://man.openbsd.org/ssh_config.5

The **command="..."** directive inserted there overrides everything else and ensures that when logging in with just that key only the script /**usr**/**local**/**bin**/**somescript.sh** is run. If it is necessary to pass parameters to the script, have a look at the contents of the **SSH_ORIGINAL_COMMAND** environment variable and use it in a case statement. Do not ever trust the contents of that variable nor use the contents directly, always indirectly.

Single-purpose keys are useful for allowing only a tunnel and nothing more. The following key will only echo some text and then exit, unless used non-interactively with the **-N** option.

```
$ grep '^command' ~/.ssh/authorized_keys
command="/bin/echo do-not-send-commands" ssh-ed25519
 AAAAC3NzaC1lZDI1NTE5AAAAIBzTIWCaILN3tHx5WW+PMVDc7DfPM9xYNY61JgFmBGrA
```

No matter what the user tries while logging in with that key, the session will only echo the given text and then exits. Using the **-N** option disables running the remote program, allowing the connection to stay open, allowing a tunnel.

```
$ ssh -L 3306:localhost:3306 \
    -i ~/.ssh/tunnel_ed25519 \
    -N \
    -l fred \
    server.example.com
```

That creates a tunnel and stays connected despite a key configuration which would close an interactive session. See also the **-n** or **-f** option for ssh(1)[47].

18.2.1 Single-purpose Keys to Avoid Remote Root Access

The easy way is to write a short shell script, place it /**usr**/**local**/**bin**/, and then configure **sudoers** to allow the otherwise unprivileged account to run just that script and only that script.

```
%wheel  ALL=(root:root) NOPASSWD: /usr/sbin/service httpd stop
%wheel  ALL=(root:root) NOPASSWD: /usr/sbin/service httpd start
```

Then the key calls the script using **command="..."** inside **authorized_keys**. Here the one key starts the web server, the other stops the web server.

```
$ grep '^command' ~/.ssh/authorized_keys
command="/usr/bin/sudo /usr/sbin/service httpd stop" ssh-ed25519
 AAAAC3NzaC1lZDI1NTE5AAAAIEcgdzDvSebOEjuegEx4W1I/aA7MM3owHfMr9yg2WH8H
command="/usr/bin/sudo /usr/sbin/service httpd start" ssh-ed25519
 AAAAC3NzaC1lZDI1NTE5AAAAIMidyqZ6OCvbWqA8Zn+FjhpYE6NoWSxVjFnFUk6MrNZ4
```

Complicated programs like rsync(1)[48], tar(1)[49], mysqldump(1)[50], and so on require an advanced approach when building a single-purpose key. For them, the **-v** option can show exactly what is being passed to the server so that **sudoers** can be set up correctly. That

47 http://man.openbsd.org/ssh.1
48 http://linux.die.net/man/1/rsync
49 http://man.openbsd.org/tar.1
50 http://linux.die.net/man/1/mysqldump

way they can be restricted to only access designated parts of the file system. For example, here is what `ssh -v` shows from one particular usage of rsync(1)[51], note the "Sending command" line:

```
$ rsync -e 'ssh -v' fred@server.example.org:/etc/ ./backup/etc/
. . .
debug1: Sending command: rsync --server --sender -e.LsfxC . /etc/
. . .
```

That output can then be added to **sudoers** so that the key can do only that function.

```
%backup ALL=(root:root) NOPASSWD: /usr/bin/rsync --server --sender -e.LsfxC .
 /etc/
```

Then to tie it all together, the account "backup" needs a key:

```
$ grep '^command' ~/.ssh/authorized_keys
command="/usr/bin/rsync --server --sender -e.LsfxC . /etc/" ssh-ed25519
 AAAAC3NzaC1lZDI1NTE5AAAAIMm0rs4eY8djqBb3dIEgbQ8lmdlxb9IAEuX/qFCTxFgb
```

Many of these programs have a —**dry-run** or equivalent option. Remember to use it when figuring out the right settings.

18.2.2 Read-only Access to Keys

In some cases it is necessary to prevent accounts from being able to changing their own authentication keys. However, such situations may be a better case for using certificates. However, if done with keys it is accomplished by putting the key file in an external directory where the user has read-only access, both to the directory and to the key file. Then the **AuthorizedKeysFile** directive assigns where sshd(8)[52] looks for the keys and can point to a secured location for the keys instead of the default location.

A good alternate location could be a new directory **/etc/ssh/authorized_keys** which could store the selected accounts' key files there. The change can be made to apply to only a group of accounts by putting the settings under a **Match** directive. The default location for keys on most systems is usually **~/.ssh/authorized_keys**.

```
Match Group sftpusers
    AuthorizedKeysFile /etc/ssh/authorized_keys/%u
```

Then the permissions there would allow the keys to be read but not written:

```
$ ls -dhln /etc/ssh/
drwxr-x--x 3 0 0 4.0K Mar 30 22:16 /etc/ssh/authorized_keys/

$ ls -dhln /etc/ssh/*.pub
-rw-r--r-- 1 0 0 173 Mar 23 13:34 /etc/ssh/fred
-rw-r--r-- 1 0 0  93 Mar 23 13:34 /etc/ssh/user1
-rw-r--r-- 1 0 0 565 Mar 23 13:34 /etc/ssh/user2
. . .
```

51 http://linux.die.net/man/1/rsync
52 http://man.openbsd.org/sshd.8

The keys could even be in within subdirectories, though the same restrictions apply regarding permissions and ownership.

For chrooted SFTP, the method is the same to keep the key files out of reach of the accounts:

```
Match Group sftpusers
    ChrootDirectory /home
    ForceCommand internal-sftp -d %u
    AuthorizedKeysFile /etc/ssh/authorized_keys/%u
```

Of course a **Match** directive is not essential. The settings could be made to apply to all accounts by putting the directive in the main part of the server configuration file instead.

18.3 Mark Public Keys as Revoked

Keys can be revoked. Keys that have been revoked can be stored in **/etc/ssh/revoked_keys**, a file specified in sshd_config(5)[53] using the directive **RevokedKeys**, so that sshd(8)[54] will prevent attempts to log in with them. No warning or error on the client side will be given if a revoked key is tried. Authentication will simply progress to the next key or method.

The revoked keys file should contain a list of public keys, one per line, that have been revoked and can no longer be used to connect to the server. The key cannot contain any extras, such as login options[55] or it will be ignored. If one of the revoked keys is tried during a login attempt, the server will simply ignore it and move on to the next authentication method. An entry will be made in the logs of the attempt, including the key's fingerprint. See the section on logging[56] for a little more on that.

```
RevokedKeys /etc/ssh/revoked_keys
```

The **RevokedKeys** configuration directive is not set in sshd_config(5)[57] by default. It must be set explicitly if it is to be used. This is another situation that might be better fulfilled through using certificate since a validity interval can be set in any combination of seconds, minutes, hours, days, or weeks can be set for certificates while keys are valid indefinitely.

18.3.1 Key Revocation Lists

A Key Revocation List (KRL) is a compact, binary form of representing revoked keys and certificates. In order to use a KRL, the server's configuration file must point to a valid list using the **RevokedKeys** directive. KRLs themselves are generated with ssh-keygen(1)[58] and can be created from scratch or edited in place. Here a new one is made, populated with a single public key:

53 http://man.openbsd.org/ssh_config.5
54 http://man.openbsd.org/sshd.8
55 https://en.wikibooks.org/wiki/OpenSSH/Client_Configuration_Files#Available_key_login_options
56 https://en.wikibooks.org/wiki/OpenSSH/Logging_and_Troubleshooting
57 http://man.openbsd.org/ssh_config.5
58 http://man.openbsd.org/ssh-keygen.1

```
$ ssh-keygen -kf  /etc/ssh/revoked_keys -z 1 ~/.ssh/old_key_rsa.pub
```

Here an existing KRL is updated by adding the **-u** option:

```
$ ssh-keygen -ukf /etc/ssh/revoked_keys -z 2 ~/.ssh/old_key_dsa.pub
```

Once a KRL is in place, it is possible to test if a specific key or certificate is in the revocation list.

```
$ ssh-keygen -Qf  /etc/ssh/revoked_keys ~/.ssh/old_key_rsa.pub
```

Only public keys and certificates will be loaded into the KRL. Corrupt or broken keys will not be loaded and will produce an error message if tried. Like with the regular **RevokedKeys** list, the public key destined for the KRL cannot contain any extras like login options or it will produce an error when an attempt is made to load it into the KRL or search the KRL for it.

18.4 Verify a Host Key by Fingerprint

The above examples have been about using keys to authenticate the client to the server. A different context in which keys are used is when the server identifies itself to the client, which happens automatically at the beginning of each non-multiplexed session. In order for that identification to happen the client acquires a public key from the server, usually on or prior to first contact, which it can subsequently use to ensure that it is connecting to the same server again and not an impostor. The default locations for storing these acquired host keys on the client are in **/etc/ssh/ssh_known_hosts**, if managed by the system administrator, or in **~/.ssh/known_hosts** if managed by the client's own account. The format of the contents is a line with a host address and its matching public key. The file is described in detail in the sshd(8)[59] manual page in the section "SSH_KNOWN_HOSTS FILE FORMAT".

When connecting for the first time to a remote host, the server's host key should be verified in order to ensure that the client is connecting to the right machine and not an impostor or anything else. Usually this verification is done by comparing the fingerprint of the server's host key rather than trying to compare the whole key itself. By default the client will show the fingerprint if the key is not already found in the **known_hosts** register.

```
$ ssh -l fred server.example.org
The authenticity of host 'server.example.org (192.0.32.10)' can't be
 established.
ECDSA key fingerprint is SHA256:LPFiMYrrCYQVsVUPzjOHv+ZjyxCHlVYJMBVFerVCP7k.
Are you sure you want to continue connecting (yes/no)?
```

That can be compared to a fingerprint received out of band, say by post, e-mail, SMS, courier, and so on. Specifically, the example represents the key's fingerprint as a base64 encoded SHA256 checksum. That is the default style. The fingerprint can also be dis-

59 http://man.openbsd.org/sshd.8

played as an MD5 hash in hexadecimal instead by passing the client's **FingerprintHash** configuration directive as a runtime argument or setting it in ssh_config(5)[60].

```
$ ssh -o FingerprintHash=md5 host.example.org
The authenticity of host 'host.example.org (192.0.32.203)' can't be established.
RSA key fingerprint is MD5:10:4a:ec:d2:f1:38:f7:ea:0a:a0:0f:17:57:ea:a6:16.
Are you sure you want to continue connecting (yes/no)?
```

But the default in new versions is SHA256 in base64 has a lower chance of collision.

In OpenSSH 6.7 and earlier, the client showed fingerprints as a hexadecimal MD5 checksum instead a of the base64-encoded SHA256 checksum currently used:

```
$ ssh -l fred server.example.org
The authenticity of host 'server.example.org (192.0.32.10)' can't be
 established.
RSA key fingerprint is 4a:11:ef:d3:f2:48:f8:ea:1a:a2:0d:17:57:ea:a6:16.
Are you sure you want to continue connecting (yes/no)?
```

Another way of comparing keys is to use the ASCII art visual host key. See further below about that.

18.4.1 Downloading keys

Even though a host's key is usually displayed for review the first time the SSH client tries to connect, it can also be fetched on demand at any time using ssh-keyscan(1)[61]:

```
$ ssh-keyscan host.example.org
# host.example.org SSH-2.0-OpenSSH_8.2
host.example.org ecdsa-sha2-nistp256 AAAAE2VjZHNhLXNoYTItbmlzdHAyNTYAAAAIbmlzdHA
yNTYAAABBBLC2PpBn
FrbXh2YoKO30Y5JdglqCWfozNiSMjsbWQt1QS09TcINqWK1aLOsNLByBE2WBymtLJEppiUVOFFPze+I=
# host.example.org SSH-2.0-OpenSSH_8.2
host.example.org ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAABAQC9iViojCZkcpdLju7/3+OaxKs
/11TAU4SuvIPTvVYvQO32o4KOdw54fQmd8f4qUWU59EUks9VQNdqf1uT1LXZN+3zXU51mCwzMzIsJuEH0
nXECtUrlpEOMlhqYh5UVkOvmOpqx1jbBVOQaTyDBOhvZsNmzp2o8ZKRSLCt9kMsEgzJmexMOHo7v3/zHe
HSD7elP7TKOJOATwqi4f6R5nNWaR6v/oNdGDtFYJnQfKUn2pdD30VtOKgUl2Wz9xDNMKrIkiM8V
sg8ly35WEuFQ1xLKjVlWSS6Frl5wLqmU1oIgowwWv+3kJS2/CRlopECy726oBgKzNoYfDOBAAbahSK8R
# host.example.org SSH-2.0-OpenSSH_8.2
host.example.org ssh-ed25519
 AAAAC3NzaC1lZDI1NTE5AAAAIDDOmBOknpyJ61Qnaeq2s+pHOH6rdMn09iREz2A/yO2m
```

Once a key is acquired, its fingerprint can be shown using ssh-keygen(1)[62]. This can be done directly with a pipe.

```
$ ssh-keyscan host.example.org | ssh-keygen -lf -
# host.example.org SSH-2.0-OpenSSH_8.2
# host.example.org SSH-2.0-OpenSSH_8.2
# host.example.org SSH-2.0-OpenSSH_8.2
256 SHA256:sxh5i6KjXZd8c34mVTBfWk6/q5cC6BzR6Qxep5nBMVo host.example.org (ECDSA)
3072 SHA256:hlPei3IXhkZmo+GBLamiiIaWbeGZMqeTXg15R42yCC0 host.example.org (RSA)
256 SHA256:ZmS+IoHh31CmQZ4NJjv3z58Pfa0zMaOgxu8yAcpuwuw host.example.org
 (ED25519)
```

60 http://man.openbsd.org/ssh_config.5
61 http://man.openbsd.org/ssh-keyscan.1
62 http://man.openbsd.org/ssh-keygen.1

If there is more than one public key type is available from the server on the port polled, then ssh-keyscan(1)[63] will fetch each of them. If there is more than one key fed via **stdin** or a file, then ssh-keygen(1)[64] will process them in order. Prior to OpenSSH 7.2 manual fingerprinting was a two step process, the key was read to a file and then processed for its fingerprint.

```
$ ssh-keyscan -t ed25519 host.example.org > key.pub
# host.example.org SSH-2.0-OpenSSH_6.8
$ ssh-keygen -lf key.pub
256 SHA256:ZmS+IoHh31CmQZ4NJjv3z58Pfa0zMaOgxu8yAcpuwuw host.example.org
 (ED25519)
```

Note that some output from ssh-keyscan(1)[65] is sent to **stderr** instead of **stdout**.

A hash, or fingerprint, can be generated manually with awk(1)[66], sed(1)[67] and xxd(1)[68], on systems where they are found.

```
$ awk '{print $2}' key.pub | base64 -d | md5sum -b | sed 's/../&:/g; s/: .*$//'
$ awk '{print $2}' key.pub | base64 -d | sha256sum -b | sed 's/ .*$//' | xxd -r
 -p | base64
```

It is possible to find all hosts from a file which have new or different keys from those in **known_hosts**, if the host names are in clear text and not stored as hashes.

```
$ ssh-keyscan -t rsa,ecdsa -f ssh_hosts | \
  sort -u - ~/.ssh/known_hosts | \
  diff ~/.ssh/known_hosts -
```

Using ssh-keyscan(1) with ssh_config(5)

The utility ssh-keyscan(1)[69] does not parse ssh_config(5)[70]. That is in part to keep the code base simple. There are a lot of configuration options which would be complicated to implement, including but not limited to **ProxyJump**, **ProxyCommand**, **Match**, **BindInterface**, and **CanonicalizeHostname**[53] . Resolving host names via the client configuration file can be done by wrapping the utility in a short shell function:

```
my-ssh-keyscan() {
    for host in "$@" ; do
        ssh-keyscan $(ssh -G "$host" | awk '/^hostname/ {print $2}')
    done
}
```

63 http://man.openbsd.org/ssh-keyscan.1
64 http://man.openbsd.org/ssh-keygen.1
65 http://man.openbsd.org/ssh-keyscan.1
66 http://linux.die.net/man/1/awk
67 http://linux.die.net/man/1/sed
68 http://linux.die.net/man/1/xxd
69 http://man.openbsd.org/ssh-keyscan.1
70 http://man.openbsd.org/ssh_config.5

That shell function uses the **-G** option of ssh(1)[71] to resolve each host name using ssh_config(5)[72] and then check the resulting host name for SSH keys.

18.4.2 ASCII Art Visual Host Key

An ASCII art representation of the key can be displayed along with the SHA256 base64 fingerprint:

```
$ ssh-keygen -lvf key
256 SHA256:BClQBFAGuz55+tgHM1aazI8FUo8eJiwmMcqg2U3UgWU www.example.org (ED25519)
+--[ED25519 256]--+
|o+=*++Eo         |
|+o .+.o.         |
|B=.oo.  .        |
|*B.=.o .         |
|= B *   S        |
|. .@ .           |
| +..B            |
|  *. o           |
| o.o.            |
+----[SHA256]-----+
```

In OpenSSH 6.7 and earlier the fingerprint is in MD5 hexadecimal form.

```
$ ssh-keygen -lvf key
2048 37:af:05:99:e7:fb:86:6c:98:ee:14:a6:30:06:bc:f0 www.example.net (RSA)
+--[ RSA 2048]----+
|          o      |
|         o .     |
|        o o      |
|       o +       |
|   .  . S        |
|    E ..         |
|  .o.* ..        |
|  .*=.+o         |
|  ..==+.         |
+-----------------+
```

18.5 More on Verifying SSH Keys

Keys on the client or the server can be verified against known good keys by comparing the base64-encoded SHA256 fingerprints.

18.5.1 Verifying Stray Client Keys

Sometimes is is necessary to compare two uncertain key files to check if they are part of the same key pair. However, public keys are more or less disposable. So the easy way in such situations on the client machine is to just rename or erase the old, problematic, public key and replace it with a new one generated from the existing private key.

71 http://man.openbsd.org/ssh.1
72 http://man.openbsd.org/ssh_config.5

```
$ ssh-keygen -y -f ~/.ssh/my_key_rsa
```

But if the two parts must really be compared, it is done in two steps using ssh-keygen(1)[73]. First, a new public key is re-generated from the known private key and used to make a fingerprint to **stdout**. Next, the fingerprint of the unknown public key is generated for comparison. In this example, the private key **my_key_a_rsa** and the public key **my_key_b_rsa.pub** are compared:

```
$ ssh-keygen -y -f my_key_a_rsa | ssh-keygen -l -f -
$ ssh-keygen -l -f my_key_b_rsa.pub
```

The result is a base64-encoded SHA256 checksum for each key with the one fingerprint displayed right below the other for easy visual comparison. Older versions don't support reading from **stdin** so an intermediate file will be needed then. Even older versions will only show an MD5 checksum for each key. Either way, automation with a shell script is simple enough to accomplish but outside the scope of this book.

18.5.2 Verifying Server Keys

Reliable verification of a server's host key must be done when first connecting. It can be necessary to contact the system administrator who can provide it out of band so as to know the fingerprint in advance and have it ready to verify the first connection.

Here is an example of the server's RSA key being read and its fingerprint shown as SHA256 base64:

```
$ ssh-keygen -lf /etc/ssh/ssh_host_rsa_key.pub
3072 SHA256:hlPei3IXhkZmo+GBLamiiIaWbeGZMqeTXg15R42yCC0 root@server.example.net
 (RSA)
```

And here the corresponding ECDSA key is read, but shown as an MD5 hexadecimal hash:

```
$ ssh-keygen -E md5 -lf /etc/ssh/ssh_host_ecdsa_key.pub
256 MD5:ed:d2:34:b4:93:fd:0e:eb:08:ee:b3:c4:b3:4f:28:e4 root@server.example.net
 (ECDSA)
```

Prior to 6.8, the fingerprint was expressed as an MD5 hexadecimal hash:

```
$ ssh-keygen -lf /etc/ssh/ssh_host_rsa_key.pub
2048 MD5:e4:a0:f4:19:46:d7:a4:cc:be:ea:9b:65:a7:62:db:2c root@server.example.net
 (RSA)
```

It is also possible to use ssh-keyscan(1)[74] to get keys from an active SSH server. However, the fingerprints still needs to be verified out of band.

73 http://man.openbsd.org/ssh-keygen.1
74 http://man.openbsd.org/ssh-keyscan.1

Warning: Remote Host Identification Has Changed!

If a server's key does not match what the client finds has been recorded in either the system's or the local account's **authorized_keys** files, then the client will issue a warning along with the fingerprint of the suspicious key.

```
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@    WARNING: REMOTE HOST IDENTIFICATION HAS CHANGED!     @
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
IT IS POSSIBLE THAT SOMEONE IS DOING SOMETHING NASTY!
Someone could be eavesdropping on you right now (man-in-the-middle attack)!
It is also possible that a host key has just been changed.
The fingerprint for the RSA key sent by the remote host is
SHA256:GkoIDP/dOI6KA9IQyOB9iqL+Rzpxx9LhlSJPCEfjVQ4.
Please contact your system administrator.
Add correct host key in /home/fred/.ssh/known_hosts to get rid of this message.
Offending RSA key in /home/fred/.ssh/known_hosts:19
  remove with:
  ssh-keygen -f "/home/fred/.ssh/known_hosts" -R "server.example.com"
RSA host key for server.example.com has changed and you have requested strict
 checking.
Host key verification failed.
```

Three reasons for the warning are common.

One reason is that the server's keys were replaced, often because the server's operating system was reinstalled without backing up the old keys. Another reason can be when the system administrator has phased out deprecated or compromised keys. However that can be planned better and if there is time to plan the migration, new keys can just be added to the server and have the clients use the **UpdateHostKeys** option so that the new keys are accepted if the old keys match. A third situation is when the connection is made to the wrong machine, such as when the remote system changes IP addresses because of dynamic address allocation.

In all three cases where the key has changed there is only one thing to do: contact the system administrator and verify the key. Ask if the OpenSSH-server was recently reinstalled, or was the machine restored from an old backup? Keep in mind that the system administrator may be you yourself in some cases.

The case which is rather rare but serious enough that it should be ruled out for sure is that the wrong machine is part of a man-in-the-middle attack.

In all four cases, an authentic key fingerprint can be acquired by any method where it is possible to verify the integrity and origin of the message, for example via PGP-signed e-mail. If physical access is possible, then use the console to get the right fingerprint. Once the authentic key fingerprint is available, return to the client machine where you got the error and remove the old key from **~/.ssh/known_hosts**

```
$ ssh-keygen -R server.example.org
```

Then try logging in, but compare the key fingerprints first and proceed if and **only** if the key fingerprint matches what you received out of band. If the key fingerprint matches, then go through with the login process and the key will be automatically added. If the key fingerprint does not match, stop immediately and figure out what you are connecting to.

It would be a good idea to get on the phone, a real phone not a computer phone, to the remote machine's system administrator or the network administrator.

18.5.3 Multiple Keys for a Host, Multiple Hosts for a Key in known_hosts

Multiple host names or IP addresses can use the same key in the **known_hosts** file by using pattern matching or simply by listing multiple systems for the same key. That can be done in either the global list of keys in **/etc/ssh/ssh_known_hosts** and the local, account-specific lists of keys in each account's **~/.ssh/known_hosts** file. Labs, computational clusters, and similar pools of machines can make use of keys in that way. Here is a key shared by three specific hosts, identified by name:

```
server1,server2,server3 ssh-rsa
 AAAAB097y0yiblo97gvl...jhvlhjgluibp7y807t08mmniKjug...==
```

Or a range can be specified by using globbing to a limited extent in either **/etc/ssh/ssh_known_hosts** or **~/.ssh/known_hosts**.

```
172.19.40.* ssh-rsa AAAAB097y0yiblo97gvl...jhvlhjgluibp7y807t08mmniKjug...==
```

Conversely, for multiple keys for the same address, it is necessary to make multiple entries in either **/etc/ssh/ssh_known_hosts** or **~/.ssh/known_hosts** for each key.

```
server1 ssh-rsa AAAAB097y0yiblo97gvljh...vlhjgluibp7y807t08mmniKjug...==
server1 ssh-rsa AAAAB0liuouibl kuhlhlu...qerf1dcw16twc61c6cw1ryer4t...==
server1 ssh-rsa AAAAB568ijh68uhg63wedx...aq14rdfcvbhu865rfgbvcfrt65...==
```

Thus in order to get a pool of servers to share a pool of keys, each server-key combination must be added manually to the **known_hosts** file:

```
server1 ssh-rsa AAAAB097y0yiblo97gvljh...07t8mmniKjug...==
server1 ssh-rsa AAAAB0liuouibl kuhlhlu...qerfw1ryer4t...==
server1 ssh-rsa AAAAB568ijh68uhg63wedx...aq14rvcfrt65...==

server2 ssh-rsa AAAAB097y0yiblo97gvljh...07t8mmniKjug...==
server2 ssh-rsa AAAAB0liuouibl kuhlhlu...qerfw1ryer4t...==
server2 ssh-rsa AAAAB568ijh68uhg63wedx...aq14rvcfrt65...==
```

Though upgrading to certificates might be a more appropriate approach that manually updating lots of keys.

18.5.4 Another way of Dealing with Dynamic (roaming) IP Addresses

It is possible to manually point to the right key using **HostKeyAlias** either as part of ssh_config(5)[75] or as a runtime parameter. Here the key for machine *Foobar* is used to connect to host 192.168.11.15

```
$ ssh -o StrictHostKeyChecking=accept-new \
      -o HostKeyAlias=foobar   \
      192.168.11.15
```

75 http://man.openbsd.org/ssh_config.5

This is useful when DHCP is not configured to try to keep the same addresses for the same machines over time or when using certain stdio forwarding methods to pass through intermediate hosts.

18.5.5 Hostkey Update and Rotation in known_hosts

A protocol extension to rotate weak public keys out of **known_hosts** has been in OpenSSH from version 6.8[54] and later. With it the server is able to inform the client of all its host keys and update **known_hosts** with new ones when at least one trusted key already known. This method still requires the private keys be available to the server [55] so that proofs can be completed. In ssh_config(5)[76], the directive **UpdateHostKeys** specifies whether the client should accept updates of additional host keys from the server after authentication is completed and add them to **known_hosts**. A server can offer multiple keys of the same type for a period before removing the deprecated key from those offered, thus allowing an automated option for rotating keys as well as for upgrading from weaker algorithms to stronger ones. See also RFC 4819: Secure Shell Public Key Subsystem[77] about key management standards.

76 http://man.openbsd.org/ssh_config.5
77 https://datatracker.ietf.org/doc/html/rfc4819

19 Certificate-based Authentication

Certificates are keys which have been signed by another key[56]. The key used for such signing is called the certificate authority. It is made in advance and set aside, reserved for signing only. Other parties use the signing key's public half to verify the authenticity of the signed key being used for server identification, in the case of a host certificate[57], or for login, in the case of a user certificate[58] .

In the interest of privilege separation, make separate certificate authorities for host certificates and user certificates if both are going to be used. As of the time of this writing, either of the elliptical curve algorithms are a good choice.

19.1 Overview of SSH Certificates

X.509[1] is an ITU Telecommunications Standardization Sector standard which defines the format of public key certificates. When using certificates either the client or the server are pre-configured to accept keys which have themselves been signed by another key specially designated and set aside for just the purpose of signing the keys actually used for work. That other key is known as a certificate authority, or CA for short. As with normal keys, certificates are used to generate signatures used in authentication. However, rather than looking up the matching public key in a file, the public key is filed with a signature and the signature is used to verify the public key and then the public key is used to ensure that the negotiations are happening with a client in possession of the matching private key. So a prerequisite for using certificates is at least a passing familiarity with normal SSH. See the chapter on Public Key Authentication[2]. The mechanics of normal key-based authentication are described briefly there in the introduction. SSH uses its own simpler certificate format and not the X.509[3] certificate format.

Two of the main advantages of certificates over keys are that they can use an expiration date, or even a date range of validity, and that they eliminate need for trust-on-first-use or complicated key verification methods. Mostly they facilitate large scale deployments by easing the processes of key approval and distribution and provide a better option than copying the same host keys across multiple destinations[4].

User certificates authenticate users to their accounts on the servers. Host certificates authenticate servers to the clients, proving that the clients are connecting to the right system.

1 https://en.wikipedia.org/wiki/X.509
2 https://en.wikibooks.org/wiki/OpenSSH/Cookbook/Public_Key_Authentication
3 https://en.wikipedia.org/wiki/X.509
4 https://en.wikibooks.org/wiki/OpenSSH/Client_Configuration_Files#About_the_Contents_of_the_known_hosts_Files

The use of a **principals** field to designate users versus hosts is the main difference between host and user certificates. In host certificates, the **principals** field refers to the server names represented by the certificate. In user certificates that field refers to the accounts which are allowed to use the certificate for logging in. Additional limitations just as specific source addresses and forced commands are available for user certificates. Date and time of validity are possible for both. Host certificates and user certificates should use separate certificate authorities. For a more authoritative resource, see the "CERTIFICATES" section of ssh-keygen(1)[5].

19.2 SSH User Certificates

User certificates authenticate the user to a server or other remote device, in other words they allow people and scripts to log in. Authenticating the client to the server by means of user certificates means that the **authorized_keys** file on the server is no longer needed. Instead the user certificate, really a signed key, is checked against the certificate authority to verify if the signature is valid. If it is, then the login process can proceed. Another advantage is that the signatures can be designated as valid only for a range of dates. So in practice, it is possible to have an expiration date. User certificates are also tied to specific accounts, referred to as 'principals'. The principal(s) allowed to use the signed key must be designated or the server will not accept use of the authentication key even if it is properly signed.

It is even possible to restrict the source of incoming connections or force specific commands using options like **force-command** and **source-address** similar to normal SSH public keys. These restrictions are set at the time of signing. If those options are to be changed, the key must be resigned or else become invalid.

19.2.1 Files Associated with SSH User Certificates

There are five files associated with user certificates. The Certificate Authority must be kept safe and stored out of band. If it is lost then no new user accounts can be added to the pool and a new certificate authority must be established. If it falls into the wrong hands then an attacker could use it to pass their accounts off as legitimate users:

- Certificate Authority - a private SSH key generated for signing other keys

The next one is kept on the server itself:

- Certificate Public Key - the public component of the certificate authority

The last three files are kept on the client systems:

- Private SSH Key - the private key used for authenticating to the server or remote device
- Public SSH Key - the matching public key which has been signed by the Certificate Authority
- User Certificate - the signature made for the User Public Key using the Certificate Authority

5 https://man.openbsd.org/ssh-keygen

Optionally, the user certificate and its key can be associated permanently with a remote server or device using the | ssh_config[6] file, either globally or per-account, via the **CertificateFile** and **IdentityFile** directives.

19.2.2 Steps for Working with SSH User Certificates

There are four steps in working with user certificates. Step one, creation of a certificate authority, is done just once per pool of users or scripts. Step two, adjusting the servers' configurations, can be repeated for as many server systems or remote devices as are needed. Step three, signing the user keys, is done once per user account. Step four, deploying the user certificate, can be done for as many clients as allowed by the **principals** and any **source-address** limitations that might be included.

1. Creating a Certificate Authority

It is a very good idea to keep separate certificate authorities for hosts and users. So, in the interest of privilege separation, make a separate certificate authority for user certificates even if you already have one for host certificates.

```
$ ssh-keygen -t ed25519 -f ~/.ssh/user_ca_key \
        -C 'User Certificate Authority for *.example.com'
```

The private key created here should be kept somewhere other than the servers. However, the servers will have access to the public component so as to be able to verify the signature that will be put forth by the clients.

2. Storing the Public Component of the Certificate Authority on the Server

The server needs to be able to look up the matching certificate in order to validate the signature on user keys. That is set in the SSH daemon's configuration. Copy the host certificate to the same place the host keys are stored. Then point the OpenSSH server to the certificate authority's public component using the **TrustedUserCAKeys** directive in |sshd_config(5)[7]

```
TrustedUserCAKeys /etc/ssh/user_ca_key.pub
```

Double check to make sure the file permissions are correct.

```
$ ls -lhn /etc/ssh/user_ca_key.pub
-rw-r--r--  1 0  0   114B May  4 16:38 /etc/ssh/user_ca_key.pub
```

Then any number of accounts can be used with that certificate authority.

6 http://man.openbsd.org/ssh_config.5
7 http://man.openbsd.org/sshd_config.5

3. Processing an Existing Public Key

The users must have a key pair already made and then submit the public key component of the pair for signing. Sign it and transfer the signed copies back to the right person. Delete any artifacts of this process immediately, since extra public keys lying around can only cause clutter at best. Here a public key named **server01.ed25519.pub** has been accepted and a certificate is made with it.

```
$ ssh-keygen -s user_ca_key -I 'edcba' -z '0002' -n fred \
        server01.ed25519.pub
```

The resulting certificate will be named **server01.ed25519-cert.pub** and will have the internal ID "edcba" and an internal serial number "2". Both the ID and the serial number must be calculated externally. For successful logins the certificate's **id** and **serial** fields will be included in the log. See the section Logging and Troubleshooting[8] for more depth on the topic. It is a very good idea to list a principal for the certificate, even for user certificates. The principal listed in the certificate does need to match the account it will log in to.

Even though the public key itself is not strictly needed after that on the client side for logging in, it can be good for the client to keep. But if the public key has been lost, a new one can be regenerated from the private key, though not the other way around. When the private key is gone, it is gone. So keep a proper backup schedule. If a file exists with the name the public key should have, it had better be the public key itself or else the login attempt will fail.

The logistics for getting the public key and delivering the certificate are outside the scope of this book. But at this point, the resulting certificate should be transferred back to the person working with the key that was submitted.

4. Logging in Using an SSH User Certificate

On the client side, both the user certificate and the private key it corresponds to are needed to log in.

```
$ ssh -o CertificateFile=server01.ed25519-cert.pub -i server01.ed25519 \
        fred@server01.example.org
```

Once things are working manually a shortcut can be made using ssh_config(5)[9] on the client. Use of **IdentitiesOnly** might also be needed if an agent is used and there are multiple keys in the agent.

```
Host server01 server01.example.org
        Hostname server01.example.org
        User fred
        IdentitiesOnly yes
        IdentityFile /home/fred/.ssh/server01.ed25519
        CertificateFile /home/fred/.ssh/server01.ed25519-cert.pub
```

8 https://en.wikibooks.org/wiki/OpenSSH/Logging_and_Troubleshooting
9 http://man.openbsd.org/ssh_config.5

With those settings, running `ssh server01` on that client will try to apply both the designated key and its corresponding user certificate and designated principal.

19.3 SSH Host Certificates

One of the main uses of keys by OpenSSH is the authentication of a remote host to a client so that the client can verify that it is connecting directly to the right system and not an impostor of via an intruder in between. For that, once acknowledged, the **known_hosts** file usually keeps a copy of the public key in a register of host keys paired with host names or IP addresses.

The difficulty with host keys in general is in populating the **known_hosts** file on clients for large pools of client machines or when connecting to new systems for the first time. In a data center or lab or Internet of Things deployment, machines are always coming and going. That means new SSH host keys each time. If there are a lot of hosts involved, then that adds up to a lot of keys in the register. Using a host certificate instead, an arbitrarily large pool of hosts using the same certificate authority need only one entry in the **known_hosts** register, even as new hosts are added to the pool. By signing the host keys which a new server or device uses to identify itself, it is still possible to roll out new systems with unique keys but have them recognized correctly and safely by clients on the first try without risking the potential for a man-in-the-middle.

By using host certificates, these identifying host keys are signed and the signature can be verified against the agreed upon certificate authority, thus greatly easing the otherwise involved process of collecting and verifying the host's public key when making the first connection.

19.3.1 Files Associated with SSH Host Certificates

There are five files involved in using host certificates. Like with user certificates, the certificate authority must be kept safe. The same precautions apply as for user certificates but for hosts rather than the accounts on them:

- Certificate Authority - a private key generated for signing other keys

The next three files are kept on the server itself.

- Certificate public key - the public component of the certificate authority
- Host Public Key - the actual key that the SSH daemon uses to identify itself to the clients
- Host Certificate - the signature made for the Host Public Key using the Certificate Authority

Then on the clients, either in the client's register or the system-wide register of recognized hosts:

- known_hosts - contains a reference to the host certificate and its principals

When the clients find and use a valid host certificate, no entry for the individual host will be added to the **known_hosts** register.

19.3.2 Steps for Using SSH Host Certificates

There are four general stages in working with host certificates. Step one, creation of a certificate authority, is done just once per server or pool of servers or devices. Step two, signing the host keys, is done once per server device as is step three. Step four, configuring the clients, can be repeated for as many client machines or individual login accounts as needed.

With each step, mind the paths. There is no one-size-fits-all solution, so it will be necessary to decide where the files should go.

1. Creating a Certificate Authority

Again, a certificate authority, or CA, is just another SSH key. However rather than using it for authenticating the servers or clients directly, it is used to sign and then validate the other keys which are then actually used for authentication.

```
$ ssh-keygen -t ecdsa -f ~/.ssh/ca_key \
        -C 'Host Certficate Authority for *.example.com'
```

The private key created here should be kept somewhere other than the servers where they will be used.

The public key in this step must be distributed out of band to the clients which will then use it to verify host identities upon first connection.

2. Fetching and Signing the Host Key

Only the public key gets signed. Fetch the remote host's public host key via a reliable method, sign it, and then upload the resulting certificate to the server. The **-h** option indicates that this shall be a host certificate and the **-s** option points to the key used to do the signing. Here a copy of the host key **ssh_host_ecdsa_key.pub** has been acquired from its server and will be worked on locally:

```
$ ssh-keygen -h -s ~/.ssh/ca_key -V '+1d' -I abcd -z 00001 \
         -n server.example.com ./ssh_host_ecdsa_key.pub
Enter passphrase:
Signed host key /etc/ssh/ssh_host_ecdsa_key-cert.pub: id "abcd" serial 1 for
 server.example.com valid from 2020-05-05T09:51:00 to 2020-05-06T09:52:01
```

The validity interval set by the **-V** option is time span relative to the date and time when the key signed. So it would be possible to make a certificate valid for only an hour tomorrow using the formula `-V '+1d2h:+1d3h'`. If no start time is set, then the value is interpreted as the stopping time. If a specific stopping time or date is required, that is best done by having a script calculate that and then call ssh_key-gen(1)[10]. If no time is given at all, the certificate will be considered valid indefinitely.

The **-I** option assigns a label for the purpose of identifying the certificate.

10 `http://man.openbsd.org/ssh-keygen.1%7C`

The **-z** option manually assigns a serial number to the certificate. That serial number must be extracted from the old certificate and then incremented if it is to be kept in sequence. The default is to not have a serial number. The **-n** option assigns a set of principals, that would be which hosts may use the certificate in the context of host certificates.

The contents of the certificate can be reviewed using the **-L** option.

```
$ ssh-keygen -L -f ssh_host_ecdsa_key-cert.pub
ssh_host_ecdsa_key-cert.pub:
        Type: ecdsa-sha2-nistp256-cert-v01@openssh.com host certificate
        Public key: ECDSA-CERT
 SHA256:kVSFLH5MP/3uJWU57JxD8xVFs7ia8Pww8/ro+pq4S50
        Signing CA: ECDSA SHA256:INewUSvbnfVbgUhtLBhh+XKL0uN99qbXjsi0jvD/IGI
 (using ecdsa-sha2-nistp256)
        Key ID: "abcd"
        Serial: 1
        Valid: from 2020-05-05T09:51:00 to 2020-05-06T09:52:01
        Principals:
                server.example.com
        Critical Options: (none)
        Extensions: (none)
```

The certificate for the public host key must be transferred over to where the server can use it. That usually means the same directory where the regular public host key is found, which is/**etc**/**ssh**/ by default. Check to be sure that the certificate has the right permissions after copying it in place.

```
$ ls /etc/ssh/ssh_host_ecdsa_key*.pub
ls -nlh /etc/ssh/ssh_host_ecdsa_key*.pub
-rw-r--r--  1 0  0   653 May  4 16:49 /etc/ssh/ssh_host_ecdsa_key-cert.pub
-rw-r--r--  1 0  0   172 Feb 21 16:09 /etc/ssh/ssh_host_ecdsa_key.pub
```

3. Publishing the Host Certificate.

Once the host certificate is in place, the SSH daemon on the remote host must be pointed at the host certificate. See sshd_config(5)[11] for more.

```
HostCertificate /etc/ssh/ssh_host_ecdsa_key-cert.pub
```

The SSH daemon must then be instructed to reload its configuration file. The exact method varies from sytem to system but ultimately the daemon will receive a HUP signal.

4. Updating Clients to Acknowledge the Designated Certificate Authority

Finally add a reference to the certificate authority in the client's **known_hosts** file:

```
@cert-authority *.example.com ecdsa-sha2-nistp256 AAAAE2VjZHNhLXNoYTItbmlzdHAyNT
YAAAAIbmlzdHAyNTYAAABBBFJQHK0u
oOpBfynyKrjF/SjsLMFewUAihosD6UL3/HkaFPI1n3XAg9D7xePyUWf8thR2e0QVl5TeGLdFiGyCgt0=
```

That's it. However, it is important to realize that each of the above steps will fail more or less silently and the client will fall back to the usual approach of verifying the host's

11 http://man.openbsd.org/sshd_config.5

identity without reporting any error for its part. Any useful debugging information will be in the daemon's logs and even that will be of limited nature. See the chapter on Logging and Troubleshooting[12] for examples.

19.4 Orchestration of Certificate Deployment

Automatically deploying certificates, either user certificates or host certificates, is beyond the scope of this book. There are many ways to do it and many factors involved. The details depend not only on which orchestration software is used, but also which specific distro or operating system is in place at the end points. In short, figure out how to do it manually and then figure out how to automate that process using the work flow allowed by the deployment scripts or software.

However, it is important to note that the certificate authority can be kept in an agent. Investigate about the **-U** option for signing.

19.5 Limiting User Certificates

Various limitations can be bound to a user certificate inside the certificate itself. These are mostly specified using the **-O** option when signing the key and include a validity interval, a forced command, source address ranges, disabling pseudo-terminal allocation, and others. See the manual page for ssh-keygen(1)[13] for an authoritative list. These limitations can be combined or used separately. The examples below will address them separately for clarity.

19.5.1 Time Limitations for User Certificates

The certificate can be made valid for a planned time period, called here as a validity interval. The validity interval can have a specific starting date and time, ending date and time, or both. However, each certificate may have only a single validity interval.

Validity intervals are specified with the **-V** option and can use an absolute date and time range or a relative one. Using the key example from the user certificate section above, the following would limit when the certificate would be valid. Specifically, it is good for a five minute period on June 24, 2020 from 1:55pm through 2pm.

```
$ ssh-keygen -V '202006241355:202006241400' \
        -s user_ca_key -I 'edcba' -z '0003' -n fred \
        server01.ed25519.pub
```

Be sure to look closely at the resulting output to ensure that the range is what it needs to be.

Relative intervals can be used, too. Here is a certificate which is good for only five minutes, starting right away:

12 https://en.wikibooks.org/wiki/OpenSSH/Logging_and_Troubleshooting
13 http://man.openbsd.org/ssh-keygen.1

```
$ ssh-keygen -V ':+5m' \
        -s user_ca_key -I 'edcba' -z '0004' -n fred \
        server01.ed25519.pub
```

Note that the seconds are included and are counted from when the signing is initiated not when the passphrase is eventually entered and the signing finally completed. So if the certificate signing is initiated at 35 seconds past the top of the minute, the expiration time will also be at 35 seconds past the fifth minute. And, again, look closely at the resulting output.

19.5.2 Forced Commands with User Certificates

A user certificate can be tied to a specific command on the server by using the **-O** option as it is created. In this example, the certificate which only ever show the time and date whenever it is used to connect to the SSH server:

```
$ ssh-keygen -O force-command='/bin/date +"%T %F"' \
        -s user_ca_key -I 'edcba' -z '0005' -n fred \
        server01.ed25519.pub
```

If there is a forced command in both the certificate and sshd_config(5)[14], then the latter takes precedence. Any command that was passed as a run time argument is overridden, yet can be found in the **SSH_ORIGINAL_COMMAND** environment variable. Commands that come from inside the certificate won't affect the **SSH_ORIGINAL_COMMAND** variable and will have to be parsed from the certificate itself, which will be held in the ephemeral file pointed to by the **SSH_USER_AUTH** environment variable.

```
$ awk '/^publickey/ {print $2,$3}' ${SSH_USER_AUTH} \
        | ssh-keygen -Lf -
```

The file will only exist while the session is still open. The **SSH_USER_AUTH** variable itself will only be set if the SSH server's configuration has **ExposeAuthInfo** set to 'yes' and the default is 'no'.

19.5.3 Source Address Restrictions on User Certificates

A certificate can be limited to a specific CIDR range.

```
$ ssh-keygen -O source-address='198.51.100.0/24,203.0.113.0/26' \
        -s user_ca_key -I 'edcba' -z '0006' -n fred \
        server01.ed25519.pub
```

The CIDR range must be valid.

14 http://man.openbsd.org/sshd_config.5

19.5.4 Viewing Limitations on User Certificates

If the certificate is at hand, it is possible to look at it in detail and see which limitations apply, certificate-side. Below is an example certificate for the account 'fred', two LAN ranges, and just over one hour of access.

```
$ ssh-keygen -Lf server01.ed25519-cert.pub

server01.ed25519-cert.pub:
        Type: ssh-ed25519-cert-v01@openssh.com user certificate
        Public key: ED25519-CERT
 SHA256:hSy7QrAApIU1LgDCUrtBK2F2TZxhvincnJ0djSDow7I
        Signing CA: ED25519 SHA256:dVgTW1INbhvHjHbeAe10R9Niu8BpejifO286RZ7/niU
 (using ssh-ed25519)
        Key ID: "edcba"
        Serial: 7
        Valid: from 2020-06-24T15:17:00 to 2020-06-24T16:23:47
        Principals:
                fred
        Critical Options:
                force-command /bin/date
 +"%T",source-address=192.168.0.0/16,10.11.9.0/26
        Extensions:
                permit-X11-forwarding
                permit-agent-forwarding
                permit-port-forwarding
                permit-pty
                permit-user-rc
```

Obviously the certificate itself cannot show any additional restrictions made server-side in the SSH server's configuration.

If the user certificate is not at hand, but is used for authentication, then limitations and all other embedded characteristics can be gleaned by using the **SSH_USER_AUTH** variable provided by the **ExposeAuthInfo** option in sshd_config(5)[15] to fetch the certificate from the server. The certificate itself will leave the **SSH_ORIGINAL_COMMAND** variable alone, so the temporary file will be the only way to see what was actually in the certificate. Again, the certificate file pointed to by the **SSH_USER_AUTH** variable will only exist while the session is open.

15 http://man.openbsd.org/sshd_config.5

20 Host-based Authentication

Host-based authentication allows hosts to authenticate on behalf of all or some of that particular host's users. Those accounts can be all of the accounts on a system or a subset designated by the **Match** directive. This type of authentication can be useful for managing computing clusters and other fairly homogeneous pools of machines.

In all, three files on the server must be prepared for host-based authentication. On the client only two must be modified, but the host itself must have SSH host keys assigned. What follows sets up host-based authentication from one system to another in a single direction. For authentication both directions, follow the procedure twice but reverse the roles of the systems.

20.1 Client-side Configurations for Host-based Authentication

On the client or source host, two files must be configured and in addition at least one host key must exist:

/etc/ssh/ssh_known_hosts - global file for public keys for known hosts

/etc/ssh/ssh_config - allow clients to request host-based authentication

Then the client's host keys must be created if they do not already exist:

/etc/ssh/ssh_host_ecdsa_key

/etc/ssh/ssh_host_ed25519_key

/etc/ssh/ssh_host_rsa_key

These three steps need to be done on each of the client systems which will be connecting to the specified host. Once set up, the accounts logged on one system will be able to connect to another system without further interactive authentication.

Note that in situations environments host-based authentication might not be considered sufficient to prevent unauthorized access since it goes on a host-by-host basis, mostly.

20.1.1 1. Populate the Client with the Server's Public Keys

The remote server's public host keys must be stored on the client system in the global client configuration file **/etc/ssh/ssh_known_hosts**. One way to get the public keys from the server is to fetch them using ssh-keyscan(1)[1] and save them.

```
$ ssh-keyscan server.example.com | tee -a /etc/ssh/ssh_known_hosts
```

Be sure to make a backup copy of **/etc/ssh/ssh_known_hosts**, if it exists, before trying anything. Another way would be to add the server's public host key to the **~/.ssh/known_hosts** files in the relevant accounts. But that way is more work.

However they are acquired and verified, the public keys listed there must obviously correspond to the private host keys on the server. Any or all of the three types can be used, RSA, ECDSA, or Ed25519. DSA should no longer be used. For verification of the public keys, refer to the section on how to verify a host key by fingerprint[2] for some relevant methods.

20.1.2 2. System-wide Client Configuration

Two changes must be made. First, the client's configuration must request host-based authentication when connecting to the designated systems. Second, the client configuration file must be set to enable ssh-keysign(8)[3]. It is a helper application to access the local host keys and generate the digital signature required for host-based authentication. Both changes can be done globally in **/etc/ssh/ssh_config**.

Here is an excerpt from **/etc/ssh/ssh_config** on the client trying host-based authentication to all machines. The **Host** directive in ssh_config(5)[4] can be used to further constrain access to a particular server or group of servers.

```
Host *.pool.example.org
    HostbasedAuthentication yes
    EnableSSHKeysign yes
```

In some distributions, such as Red Hat Enterprise Linux 8, the **EnableSSHKeysign** directive may need to be placed into the general section before it will work. In that case, do this:

```
Host *.pool.example.org
    HostbasedAuthentication yes

Host *
    EnableSSHKeysign yes
```

Other configuration directives can be added as needed. For example, here are two additional settings applied to the same pool:

1 http://man.openbsd.org/ssh-keyscan.1
2 https://en.wikibooks.org/wiki/OpenSSH/Cookbook/Public_Key_Authentication#Verify_a_Host_Key_by_Fingerprint
3 http://man.openbsd.org/ssh-keysign.8
4 http://man.openbsd.org/ssh_config.5

```
Host *.pool.example.org
    HostbasedAuthentication yes
    EnableSSHKeysign yes
    ServerAliveCountMax 3
    ServerAliveInterval 60
```

If the home directory of the client host is one that is shared with other machines, say using NFS or AFS, then it may be useful to look at the **NoHostAuthenticationForLocalhost** directive, too.

As a bit of trivia, the program ssh-keysign(8)[5] itself must be SUID root. But SUID was probably set when it was installed and no changes there are needed.

20.1.3 3. Set Up the Client System's Own Host Keys

The program ssh-keysign(8)[6] needs to read the client system's private host keys in /**etc**/**ssh**/. Installing the OpenSSH server will create these keys but if the OpenSSH server has not been installed on the client system, it does not need to be. It is enough to have the keys by themselves. If the client system's private host keys do not exist, it will be necessary to add them manually using ssh-keygen(1)[7] before host-based authentication will work.

```
$ ssh-keygen -A
```

The default path is /**etc**/**ssh**/ when using the **-A** option.

Note that although the client's system does not have to have an SSH server actually running in order to use host-based authentication to reach another system, it is entirely feasible to install but then disable or uninstall the SSH server on the client's server as a way to get the host keys in place.

20.2 Server-side Configurations for Host-based Authentication

Three files on the server or target host must be modified to enable and allow host-based authentication:

/**etc**/**shosts.equiv** same syntax as old **rhosts.equiv**

/**etc**/**ssh**/**ssh_known_hosts** holds the host public keys from the clients

/**etc**/**ssh**/**sshd_config** turns on host-based authentication

The exact location of **shosts.equiv** may vary depending on operating system and distro.

5 http://man.openbsd.org/ssh-keysign.8
6 http://man.openbsd.org/ssh-keysign.8
7 http://man.openbsd.org/ssh-keygen.1

20.2.1 1. Registering the Allowed Client Systems with the Server

The **shosts.equiv** file must contain a list of the client systems which are allowed to attempt host-based authentication. The file can contain host names, IP addresses, or net groups. It is best to keep this file simple and oriented to just the list of hosts, either by name or IP number. It provides only the first cut, anyway. For fine tuning, use sshd_config(5)[8] instead to set or revoke access for specific users and groups.

It is important to note that if you are using the **shosts.equiv file**, the user name on the client and server system must match. For instance, to connect to bob@server1.example.org, the user name on the client must also be bob. If you need user 'alice' to connect to bob@server1.example.org, you need to specify this in the **.shosts** file for 'bob', not in the global **shosts.equiv** file.

```
client1.example.org
192.0.2.102
client8.example.org -bull
@statcluster
```

This file resides in the directory /**etc**/ in OpenBSD, which this book uses for the reference system when possible. On other systems the file might be found in the directory /**etc**/**ssh**/ . Either way, the **shosts.equiv** file identifies which addresses are allowed to try authenticating. Check the manual page for sshd(1)[9] on the actual system being used to be sure of the correct location.

Hunt for a manual page **hosts.equiv(5)** for more details on **.shosts.equiv** (and **.shosts.**), but consider that its use is long since deprecated and most systems won't even have that manual page.

Leftovers from the Old Days

Two legacy files may be on really old systems. and are optionally modified if they are referred to from within **shosts.equiv**. Each line in the **netgroup** file consists of a net group name followed by a list of the members of the net group, specifically host, user, and domain.

/**etc**/**netgroup** - default netgroup list

/**etc**/**netgroup.db** - netgroup database, build from netgroup

However, these are mostly legacy from the old **rhosts** and should be avoided.

Another leftover is using **.shosts** in each account's home directory. That would be a local equivalent to **.shosts.equiv**. In that way, individual users can have a local **.shosts** containing a list of trusted remote machines, or user-machine pairs, which are allowed to try host-based authentication.

.shosts must not writable by any group or any other users. Permissions set to 0644 should do it. The usage and format of **.shosts** is exactly the same as .rhosts, but allows host-based authentication without permitting login by insecure, legacy tools **rlogin** and **rsh**. The list

8 http://man.openbsd.org/sshd_config.5
9 http://man.openbsd.org/sshd.1

is one line per host. The first column is obligatory and contains the name or address of the host permitted to attempt host-based authentication.

However a global **.shosts.equiv** is preferable to having **.shosts** in each and every home directory.

20.2.2 2. Populate the Server with the Client's Public Keys

The client systems listed in the server's **shosts.equiv** must also have their public keys in **/etc/ssh/ssh_known_hosts** on the server in order to be acknowledged. There are three required data fields per line. First is the host name or IP address or comma separated list of them, corresponding to those from **shosts.equiv**. Next is the key type, either ssh-rsa for RSA keys or ssh-ed25519 for Ed25519 keys or ecdsa-sha2-nistp256 for ECDSA keys. The third field is the public key itself. Last, and optionally, can be a comment about the key.

```
desktop,192.0.2.102 ssh-rsa AAAAB3NzaC1yc2EAAAABIw ... qqU24CcgzmM=
```

Just like step one for the client, there are many ways of collecting the public key information from the client and getting it to the server. It can be copied using sftp(1)[10], copied from **~/.ssh/known_hosts**, or grabbed using ssh-keyscan(1)[11]. Though the latter two methods only work if the client system also has an SSH server running.

```
$ ssh-keyscan -t rsa client.example.org | tee -a /etc/ssh/ssh_known_hosts
```

20.2.3 3. System-wide Server Configuration

The third file that must be changed on the server is sshd_config(5)[12]. It must be told to allow host-based authentication by setting the **HostbasedAuthentication** directive, either for all users or just some users or just certain groups.

```
HostbasedAuthentication yes
```

Host-based authentication can be limited to specific users or groups. Here is an example excerpt from sshd_config(5)[13] allowing allow any user in the group 'cluster2' to let the hosts authenticate on their behalf:

```
Match Group cluster2
        HostbasedAuthentication yes
```

Certain host keys types can be allowed using **HostbasedAcceptedAlgorithms**, formerly **HostbasedAcceptedKeyTypes**, with a comma-delimited list of acceptable key algorithms. All the key algorithms which are to be allowed must be listed because those not listed are not allowed. Patterns can be used in the whitelist. Below, Ed25519 and ECDSA keys are allowed, but others, such as RSA and DSA are not.

10 http://man.openbsd.org/sftp.1
11 http://man.openbsd.org/ssh-keyscan.1
12 http://man.openbsd.org/sshd_config.5
13 http://man.openbsd.org/sshd_config.5

```
HostbasedAuthentication yes
HostbasedAcceptedAlgorithms ssh-ed25519*,ecdsa-sha2*
```

Host Names and Other DNS Matters

Make complete DNS entries for the clients if possible, including allowing reverse lookups. If the client machine is not listed in DNS, then the server might have trouble recognizing it. In that case you might have to tell sshd(8)[14] not to do reverse lookups in DNS for connecting hosts. This can be a problem on informal LANs where hosts have addresses but no registered host names. Here is an example from **/etc/ssh/sshd_config** to work around lack of DNS records for the client using the **HostbasedUsesNameFromPacketOnly** directive.

```
HostbasedAuthentication yes
HostbasedUsesNameFromPacketOnly yes
```

Don't add this directive unless the normal way fails. Otherwise it can interfere and prevent authentication.

Sometimes the host trying to connect gets identified to the target host as something other than what was expected. That too will block authentication. So make sure that the configuration files match what the host actually calls itself.

20.3 Debugging

Configuration should be quite straight forward, with small changes in only three files on the server and two on the client to manage. If there are difficulties, be prepared to run sshd(8)[15] standalone at debug level 1 (**-d**) to 3 (**-ddd**) and ssh(1)[16] at debug level 3 (**-vvv**) a few times to see what you missed. The mistakes have to be cleared up in the right order, so take it one step at a time.

If the server produces the message "*debug3: auth_rhosts2_raw: no hosts access file exists*" turns up, the **shosts.equiv** file is probably in the wrong place or missing and no fallback **~/.shosts** lies in reserve in that account.

If the server cannot find the key for the client despite it being in **known_hosts** and if the client's host name is not in regular DNS, then it might be necessary to add the directive **HostbasedUsesNameFromPacketOnly**. This uses the name supplied by the client itself rather than doing a DNS lookup.

Here is a sample excerpt from a successful host-based authentication for user 'fred' from the host at *192.0.2.102*, also known as *desktop1*, using an Ed25519 key. The server first tries looking for an ECDSA key and does not find it.

14 http://man.openbsd.org/sshd.8
15 http://man.openbsd.org/sshd.8
16 http://man.openbsd.org/ssh.1

```
# /usr/sbin/sshd -ddd
debug2: load_server_config: filename /etc/ssh/sshd_config
...
debug3: /etc/ssh/sshd_config:111 setting HostbasedAuthentication yes
debug3: /etc/ssh/sshd_config:112 setting HostbasedUsesNameFromPacketOnly yes
...
debug1: sshd version OpenSSH_6.8, LibreSSL 2.1
...
debug1: userauth-request for user fred service ssh-connection method hostbased
 [preauth]
debug1: attempt 1 failures 0 [preauth]
debug2: input_userauth_request: try method hostbased [preauth]
debug1: userauth_hostbased: cuser fred chost desktop1. pkalg ecdsa-sha2-nistp256
 slen 100 [preauth]
...
debug3: mm_answer_keyallowed: key_from_blob: 0x76eede00
debug2: hostbased_key_allowed: chost desktop1. resolvedname 192.0.2.102 ipaddr
 192.0.2.102
debug2: stripping trailing dot from chost desktop1.
debug2: auth_rhosts2: clientuser fred hostname desktop1 ipaddr desktop1
debug1: temporarily_use_uid: 1000/1000 (e=0/0)
debug1: restore_uid: 0/0
debug1: fd 4 clearing O_NONBLOCK
debug2: hostbased_key_allowed: access allowed by auth_rhosts2
debug3: hostkeys_foreach: reading file "/etc/ssh/ssh_known_hosts"
debug3: record_hostkey: found key type ED25519 in file
 /etc/ssh/ssh_known_hosts:1
debug3: load_hostkeys: loaded 1 keys from desktop1
debug1: temporarily_use_uid: 1000/1000 (e=0/0)
debug3: hostkeys_foreach: reading file "/home/fred/.ssh/known_hosts"
debug1: restore_uid: 0/0
debug1: check_key_in_hostfiles: key for host desktop1 not found
Failed hostbased for fred from 192.0.2.102 port 10827 ssh2: ECDSA
 SHA256:CEXGTmrVgeY1qEiwFe2Yy3XqrWdjm98jKmXOLK5mlQg, client user "fred", client
 host "desktop1"
debug3: mm_answer_keyallowed: key 0x76eede00 is not allowed
debug3: mm_request_send entering: type 23
debug2: userauth_hostbased: authenticated 0 [preauth]
debug3: userauth_finish: failure partial=0 next
 methods="publickey,password,keyboard-interactive,hostbased" [preauth]
debug1: userauth-request for user fred service ssh-connection method hostbased
 [preauth]
debug1: attempt 2 failures 1 [preauth]
debug2: input_userauth_request: try method hostbased [preauth]
debug1: userauth_hostbased: cuser fred chost desktop1. pkalg ssh-ed25519 slen 83
 [preauth]
debug3: mm_key_allowed entering [preauth]
debug3: mm_request_send entering: type 22 [preauth]
debug3: mm_key_allowed: waiting for MONITOR_ANS_KEYALLOWED [preauth]
debug3: mm_request_receive_expect entering: type 23 [preauth]
debug3: mm_request_receive entering [preauth]
debug3: mm_request_receive entering
debug3: monitor_read: checking request 22
debug3: mm_answer_keyallowed entering
debug3: mm_answer_keyallowed: key_from_blob: 0x7e499180
debug2: hostbased_key_allowed: chost desktop1. resolvedname 192.0.2.102 ipaddr
 192.0.2.102
debug2: stripping trailing dot from chost desktop1.
debug2: auth_rhosts2: clientuser fred hostname desktop1 ipaddr desktop1
debug1: temporarily_use_uid: 1000/1000 (e=0/0)
debug1: restore_uid: 0/0
debug1: fd 4 clearing O_NONBLOCK
debug2: hostbased_key_allowed: access allowed by auth_rhosts2
debug3: hostkeys_foreach: reading file "/etc/ssh/ssh_known_hosts"
debug3: record_hostkey: found key type ED25519 in file
 /etc/ssh/ssh_known_hosts:1
```

```
debug3: load_hostkeys: loaded 1 keys from desktop1
debug1: temporarily_use_uid: 1000/1000 (e=0/0)
debug3: hostkeys_foreach: reading file "/home/fred/.ssh/known_hosts"
debug1: restore_uid: 0/0
debug1: check_key_in_hostfiles: key for desktop1 found at
 /etc/ssh/ssh_known_hosts:1
Accepted ED25519 public key SHA256:BDBRg/JZ36+PKYSQTJDsWNW9rAfmUQCgWcY7desk/+Q
 from fred@desktop1
debug3: mm_answer_keyallowed: key 0x7e499180 is allowed
debug3: mm_request_send entering: type 23
debug3: mm_key_verify entering [preauth]
debug3: mm_request_send entering: type 24 [preauth]
debug3: mm_key_verify: waiting for MONITOR_ANS_KEYVERIFY [preauth]
debug3: mm_request_receive_expect entering: type 25 [preauth]
debug3: mm_request_receive entering [preauth]
debug3: mm_request_receive entering
debug3: monitor_read: checking request 24
debug3: mm_answer_keyverify: key 0x7e49a700 signature verified
debug3: mm_request_send entering: type 25
Accepted hostbased for fred from 192.0.2.102 port 10827 ssh2: ED25519
 SHA256:BDBRg/JZ36+PKYSQTJDsWNW9rAfmUQCgWcY7desk/+Q, client user "fred", client
 host "desktop1"
debug1: monitor_child_preauth: fred has been authenticated by privileged process
...
```

Note any warnings or error messages and read them carefully. If there is too much output, remember the **-e** option for the server and use it to save the debugging output to a separate file and then read that file afterwards.

Since this method relies on the client as well as the server, running the client with increased verbosity can sometimes help too.

```
$ ssh -v fred@server.example.org
$ ssh -vv fred@server.example.org
$ ssh -vvv fred@server.example.org
```

If there is too much output from the client to handle, remember the **-E** option and redirect the debug logs to a file and then read that file at leisure.

21 Load Balancing

21.1 MaxStartups

Random early drop can be enabled by specifying the three colon-separated values *start*:*rate*:*full*. After the number of unauthenticated connections reaches the value specified by *start*, sshd(8)[1] will begin to refuse new connections at a percentage specified by *rate*. The proportional rate of refused connections then increases linearly as the limit specified by *full* is approached until 100% is reached. At that point all new attempts at connection are refused until the backlog goes down.

```
MaxStartups   10:30:100
```

For example, if **MaxStartups 5:30:90** is given in sshd_config(5)[2], then starting with 5 new connections pending authentication the server will start to drop 30% of the new connections. By the time the backlog increases to 90 pending unauthenticated connections, 100% will be dropped.

In the default settings, the value for *full* has been increased to 100 pending connections to make it harder to succumb to a denial of service due to attack or heavy load. So the new default is 10:30:100.

Alternately, if the number of incoming connections is not managed at the network level by a packet filter or other tricks like round-robin DNS, it is possible to limit it at the SSH server itself. Setting **MaxStartups** to an integer sets a hard limit to the maximum number of concurrent unauthenticated connections to the SSH daemon.

```
MaxStartups   10
```

Additional connections will be dropped until authentication succeeds or the **LoginGraceTime** expires for another connection. The old default was 10.

21.2 Preventing Timeouts Of A Not So Active Session

There are two connections that can be tracked during an SSH session, the network's TCP connection and the encrypted SSH session traveling on top of it. Some tunnels and VPNs might not be active all the time, so there is the risk that the session times out or even that

1 http://man.openbsd.org/sshd.8
2 http://man.openbsd.org/sshd_config.5

the TCP session times out at the router or firewall. The network connection can be be tracked with **TCPKeepAlive**, but is not an accurate indication of the state of the actual SSH connections. It is, however, a useful indicator of the status of the actual network. Either the client or the server can counter that by keeping either the encrypted connection active using a heartbeat.

On the client side, if the global client configuration is not already set, individuals can use **ServerAliveInterval** to choose an interval in seconds for server alive heartbeats and use **ServerAliveCountMax** to set the corresponding maximum number of missed client messages allowed before the encrypted SSH session is considered closed.

```
ServerAliveInterval  15
ServerAliveCountMax  4
```

On the server side, **ClientAliveInterval** sets the interval in seconds between client alive heartbeats. **ClientAliveCountMax** sets the maximum number of missed client messages allowed before the encrypted SSH session is considered closed. If no other data has been sent or received during that time, sshd(8)[3] will send a message through the encrypted channel to request a response from the client. If sshd_config(5)[4] has **ClientAliveInterval** set to 15, and **ClientAliveCountMax** set to 4, unresponsive SSH clients will be disconnected after approximately 60 (= 15 x 4) seconds.

```
ClientAliveInterval  15
ClientAliveCountMax  4
```

If a time-based **RekeyLimit** is also used but the time limit is shorter than the **ClientAliveInterval** heartbeat, then the shorter re-key limit will be used for the heartbeat interval instead.

This is more or less the same principal as with the server. Again, that is set in ~/.ssh/config[5] and can be applied globally to all connections from that account or selectively to specific connections using a **Host** or **Match** block.

21.3 Ensuring Timeouts Of An Inactive Interactive Session

If the server is no longer disconnecting idle SSH accounts when they reach the timeout configured by the **ClientAliveInterval** option, then the work-around for that is to set the shell's **TMOUT** variable to the desire timeout value. When **TMOUT** is set, it specifies the number of seconds the shell will wait for a line of input to be entered before closing the shell and thus the SSH session. Note that this means pressing *Enter*, too, because other typing will not be enough by itself to prevent the timeout and the line must actually be entered for the timer to be reset.

On the server, check for the presence of the **SSH_CONNECTION** variable, which is usually empty unless currently in an SSH session, to differentiate an SSH session from a

3 http://man.openbsd.org/sshd.8
4 http://man.openbsd.org/sshd_config.5
5 http://man.openbsd.org/ssh_config.5

local shell. If the account is allowed to make changes to the timeout, then the following can be in the account's own profile, such as **~/.profile**,

```
if [ "$SSH_CONNECTION" != "" ]; then
        # 10 minutes
        TMOUT=600
        export TMOUT
fi
```

If the account must not be able to change this setting, then it must be in the global profile and made read-only, such as somewhere under /**etc**/**profile.d**/,

```
if [ "$SSH_CONNECTION" != "" ]; then
        # 10 minutes
        TMOUT=600
        readonly TMOUT
        export TMOUT
fi
```

Both examples above are for Bourne shells, their derivatives, and maybe some other shells. A few shells might have other options available, such as an actual **autologout** variable found in tcsh(1)[6].

21.4 TCP Wrappers, Also Called tcpd(8)

As of 6.7, OpenSSH's sshd(8)[7] no longer supports TCP Wrappers, also referred to as tcpd(8)[8]. So this subsection only applies to 6.6 and earlier. Other options that can be used instead of tcpd(8)[9] include packet filters like PF [59], ipf, NFTables, or even old IPTables. In particular, the **Match** directive for the OpenSSH server supports filtering by CIDR address. Use these instead and keep in mind the phrase "equivalent security control" which can smooth out hassles caused by security auditors which may still have a "tcpwrappers" checkbox left over from the 1990s on their worksheets.

The tcpd(8)[10] program was an access control utility for incoming requests to Internet services. It was used for services that have a one-to-one mapping to executables, such as sshd(8)[11], and which have been compiled to interact with it. It checked first a whitelist (/**etc**/**hosts.allow**) and then a blacklist (/**etc**/**hosts.deny**) to approve or deny access. The first pattern to match any given connection attempt was used. The default in /**etc**/**hosts.deny** was to block access, if no rules match in /**etc**/**hosts.allow**:

```
sshd: ALL
```

6 https://linux.die.net/man/1/tcsh
7 http://man.openbsd.org/sshd.8
8 http://linux.die.net/man/8/tcpd
9 http://linux.die.net/man/8/tcpd
10 http://linux.die.net/man/8/tcpd
11 http://man.openbsd.org/sshd.8

In addition to access control, tcpd(8)[12] can be set to run scripts using twist or spawn when a rule is triggered. spawn launches another program as a child process of tcpd(8)[13]. From /**etc**/**hosts.allow**:

```
sshd: .example.org : allow
sshd: .example.com : spawn \
    /bin/date -u +"%%F %%T UTC from %h" >> /var/log/sshd.log : allow
```

The variable **%h** expands to the connecting clients host name or ip number. The manual page for hosts_access(5)[14] includes a full description of the variables available. Because the program in the example, date, uses the same symbol for variables, the escape character (%) must be escaped (%%) so it will be ignored by tcpd(8)[15] and get passed on to date correctly.

twist replaces the service requested with another program. It is sometimes used for honeypots, but can really be used for anything. From /**etc**/**hosts.deny**:

```
sshd: .example.org : deny
sshd: ALL : twist /bin/echo "Sorry, fresh out." : deny
```

With TCP Wrappers the whitelist /**etc**/**hosts.allow** is searched first, then the blacklist /**etc**/**hosts.deny**. The first match is applied. If no applicable rules are found, then access is granted by default. It should not be used any more and the better alternatives used instead. See also the **Match** directive in sshd_config(5)[16] about CIDR addresses or else the **AllowUsers** and **DenyUsers** directives.

21.4.1 Using TCP Wrappers To Allow Connections From Only A Specific Subnet

It was possible to use TCP Wrappers just set sshd(8)[17] to listen only to the local address and not accept any external connections. That was one way. To use TCP Wrappers for that, put a line in /**etc**/**hosts.deny** blocking everything:

```
sshd: ALL
```

And add an exception for the in /**etc**/**hosts.allow** by designating the ip range using CIDR notation or by domain name

```
sshd: 192.0.32.0/20
```

The same method can be used to limit access to just the localhost (127.0.0.0/8) by adding a line to /**etc**/**hosts.allow**:

```
sshd: 127.0.0.0/8
```

12 http://linux.die.net/man/8/tcpd
13 http://linux.die.net/man/8/tcpd
14 http://linux.die.net/man/5/hosts_access
15 http://linux.die.net/man/8/tcpd
16 http://man.openbsd.org/sshd_config.5
17 http://man.openbsd.org/sshd.8

Again, the best practice is to block from everywhere and then open up exceptions. Keep in mind that if domains are used instead of IP ranges, DNS entries must be in order and DNS itself accessible. However, the above is of historical interest only. The same kind of limitations are better done by setting sshd_config(5)[18] accordingly and instead of using TCP Wrappers utilize the **Match** directive in the OpenSSH server or the packet filter in the operating system itself.

21.5 The Extended Internet Services Daemon (xinetd)

The Extended Internet Services Daemon, xinetd(8)[19], can provide many kinds of access control. That includes but is not limited to name, address or network of remote host, and time of day. It can place limits on the number of services per each service as well as discontinue services if loads exceed a certain limit.

The super-server listens for incoming requests and launches sshd(8)[20] on demand, so it is necessary to first stop sshd(8)[21] from running as standalone daemon. This may mean modifying System V init scripts or Upstart configuration files. Then make an xinetd(8)[22] configuration file for the service SSH. It will probably go in **/etc/xinetd.d/ssh** The argument **-i** is important as it tells sshd(8)[23] that it is being run from xinetd(8)[24].

```
service ssh
{
    socket_type     = stream
    protocol        = tcp
    wait            = no
    user            = root
    server          = /usr/sbin/sshd
    server_args     = -i
    per_source      = UNLIMITED
    log_on_failure  = USERID HOST
    access_times    = 08:00-15:25
    banner          = /etc/banner.inetd.connection.txt
    banner_success  = /etc/banner.inetd.welcome.txt
    banner_fail     = /etc/banner.inetd.takeahike.txt

    # log_on_success  = PID HOST DURATION TRAFFIC EXIT
    # instances       = 10
    # nice            = 10
    # bind            = 192.168.0.100
    # only_from       = 192.168.0.0
    # no_access       = 192.168.54.0
    # no_access       += 192.168.33.0
}
```

Finally, reload the configuration by sending SIGHUP to xinetd(8)[25].

18 http://man.openbsd.org/sshd_config.5
19 http://linux.die.net/man/8/xinetd
20 http://man.openbsd.org/sshd.8
21 http://man.openbsd.org/sshd.8
22 http://linux.die.net/man/8/xinetd
23 http://man.openbsd.org/sshd.8
24 http://linux.die.net/man/8/xinetd
25 http://linux.die.net/man/8/xinetd

22 Multiplexing

Multiplexing is the ability to send more than one signal over a single line or connection. In OpenSSH, multiplexing can re-use an existing outgoing TCP connection for multiple concurrent SSH sessions to a remote SSH server, avoiding the overhead of creating a new TCP connection and reauthenticating each time.

22.1 Advantages of Multiplexing

An advantage of SSH multiplexing is that the overhead of creating new TCP connections and negotiating the secure connection is eliminated. The overall number of connections that a machine may accept is a finite resource and the limit is more noticeable on some machines than on others and varies greatly depending on both load and usage. There is also a significant delay when opening a new connection. Activities that repeatedly open new connections can be significantly sped up using multiplexing.

The difference between multiplexing and standalone sessions can be seen by comparing the tables below. Both are selected output from `netstat -nt` slightly edited for clarity. We see in Table 1, "SSH Connections, Separate", that without multiplexing each new login creates a new TCP connection, one per login session. Following that we see in the other table, Table 2, "SSH Connections, Multiplexed", that when multiplexing is used, new logins are channelled over the already established TCP connection.

```
#              Local Address       Foreign Address         State

# one connection
tcp    0   0   192.168.x.y:45050   192.168.x.z:22       ESTABLISHED

# two separate connections
tcp    0   0   192.168.x.y:45050   192.168.x.z:22       ESTABLISHED
tcp    0   0   192.168.x.y:45051   192.168.x.z:22       ESTABLISHED

# three separate connections
tcp    0   0   192.168.x.y:45050   192.168.x.z:22       ESTABLISHED
tcp    0   0   192.168.x.y:45051   192.168.x.z:22       ESTABLISHED
tcp    0   0   192.168.x.y:45052   192.168.x.z:22       ESTABLISHED
```

Table 1: SSH Connections, Separate

Both tables show TCP/IP connections associated with SSH sessions. The table above shows a new TCP/IP connection for each new SSH connection. The table below shows a single TCP/IP connection despite multiple active SSH sessions.

```
#              Local Address       Foreign Address         State

# one connection
tcp    0   0   192.168.x.y:58913   192.168.x.z:22       ESTABLISHED

# two multiplexed connections
tcp    0   0   192.168.x.y:58913   192.168.x.z:22       ESTABLISHED

# three multiplexed connections
tcp    0   0   192.168.x.y:58913   192.168.x.z:22       ESTABLISHED

Table 2: SSH Connections, Multiplexed
```

As we can see with multiplexing, only a single TCP connection is set up and used regardless of whether or not there are multiple SSH sessions carried over it.

Or we can compare the time it takes to run true(1)[1] on a slow remote server, using time(1)[2]. The two commands would be something like `time ssh server.example.org true` versus `time ssh -S ./path/to/somesocket server.example.org true` and use keys with an agent for each of them. First without multiplexing, we see the normal connection time:

```
real    0m0.658s
user    0m0.016s
sys     0m0.008s
```

Then we do the same thing again, but with a multiplexed connection to see a faster result:

```
real    0m0.029s
user    0m0.004s
sys     0m0.004s
```

The difference is quite large and will definitely add up for any activity where connections are made repeatedly in rapid succession. The speed gain for multiplexed connections doesn't come with the master connection, that is normal speed, but with the second and subsequent multiplexed connections. The overhead for a new SSH connection remains, but the overhead of a new TCP connection is avoided. The second and later connections will reuse the established TCP connection over and over and not need to create a new TCP connection for each new SSH connection.

22.2 Setting Up Multiplexing

The OpenSSH client supports multiplexing its outgoing connections, since version 3.9 (August 18, 2004)[60], using the **ControlMaster**, **ControlPath** and **ControlPersist** configuration directives which get defined in ssh_config(5)[3]. The client configuration file usually defaults to the location **~/.ssh/config**. All three directives are described in the manual page for ssh_config(5)[4]. See also the "TOKENS" section there for the list of tokens available for use in the **ControlPath**. Any valid tokens used are expanded at run time.

1 http://man.openbsd.org/true.1
2 http://linux.die.net/man/1/time
3 http://man.openbsd.org/ssh_config.5
4 http://man.openbsd.org/ssh_config.5

ControlMaster determines whether ssh(1)[5] will listen for control connections and what to do about them. **ControlPath** sets the location for the control socket used by the multiplexed sessions. These can be either globally or locally in ssh_config(5)[6] or else specified at run time. Control sockets are removed automatically when the master connection has ended. **ControlPersist** can be used in conjunction with **ControlMaster**. If **ControlPersist** is set to 'yes', then it will leave the master connection open in the background to accept new connections until either killed explicitly or closed with **-O** or ends at a pre-defined timeout. If **ControlPersist** is set to a time, then it will leave the master connection open for the designated time or until the last multiplexed session is closed, whichever is longer.

Here is a sample excerpt from ssh_config(5)[7] applicable for starting a multiplexed session to *machine1.example.org* via the shortcut *machine1*.

```
Host machine1
        HostName machine1.example.org
        ControlPath ~/.ssh/controlmasters/%r@%h:%p
        ControlMaster auto
        ControlPersist 10m
```

With that configuration, the first connection to *machine1* will create a control socket in the directory **~/.ssh/controlmasters/** Then any subsequent connections, up to 10 by default as set by **MaxSessions** on the SSH server, will re-use that control path automatically as multiplexed sessions. Confirmation of each new connection can be required if **ControlMaster** set to 'autoask' instead of 'auto'.

Please note with the settings above that the control socket would be placed into the folder **~/.ssh/controlmasters/**. If that folder doesn't exist first, the SSH client will exit with a specific error about unix_listener complaining that the file or path does not exist:

```
unix_listener: cannot bind to path
 /home/fred/.ssh/controlmasters/fred@machine1.example.org:22.EOWsvm5xFt30O6CB: No
 such file or directory
```

The option **-O** ssh(1)[8] can be used to manage the connection using the same shortcut configuration. To cancel all existing connections, including the master connection, use 'exit' instead of 'stop'.

```
$ ssh -O check machine1
Master running (pid=14379)
$ ssh -O stop machine1
Stop listening request sent
$ ssh -O check machine1
Control socket connect(/Users/Username/.ssh/sockets/machine1): No such file or
 directory
```

In that example, the status of the connection is checked first. Then the master connection is told not to accept further multiplexing requests and finally we check again that no control socket is available.

5 http://man.openbsd.org/ssh.1
6 http://man.openbsd.org/ssh_config.5
7 http://man.openbsd.org/ssh_config.5
8 http://man.openbsd.org/ssh.1

Manually Establishing Multiplexed Connections

Multiplexed sessions need a control master to connect to. The run time parameters **-M** and **-S** also correspond to **ControlMaster** and **ControlPath**, respectively. So first an initial master connection is established using **-M** when accompanied by the path to the control socket using **-S**.

```
$ ssh -M -S /home/fred/.ssh/controlmasters/fred@server.example.org:22
 server.example.org
```

Then subsequent multiplexed connections are made in other terminals. They use **ControlPath** or **-S** to point to the control socket.

```
$ ssh -S /home/fred/.ssh/controlmasters/fred@server.example.org:22
 server.example.org
```

Note that the control socket is named "fred@server.example.org:22", or *%r@%h:%p*, to try to make the name unique. The combination %r, %h and %p stand for the remote user name, the remote host and the remote host's port. The control sockets should be given unique names.

Multiple **-M** options place ssh(1)[9] into **master** mode with confirmation required before slave connections are accepted. This is the same as **ControlMaster=ask**. Both require X to ask for the confirmation.

Here is the master connection for the host *server.example.org* as set to ask for confirmation of new multiplexed sessions:

```
$ ssh -MM -S ~/.ssh/controlmasters/%r@%h:%p server.example.org
```

And here is a subsequent multiplexed connection:

```
$ ssh -S ~/.ssh/controlmasters/%r@%h:%p server.example.org
```

The status of the control master connection can be queried using **-O check**, which will tell if it is running or not.

```
$ ssh -O check -S ~/.ssh/controlmasters/%r@%h:%p server.example.org
```

If the control session has been stopped, the query will return an error about "No such file or directory" even if there are still multiplexed sessions running because the socket is gone.

Alternately, instead of specifying **-M** and **-S** as runtime options, configuration options can be spelled out in full using **-o** for easier transfer to the client configuration file when figured out.

The way below sets configuration options as run time parameters by first starting a control master.

```
$ ssh -o "ControlMaster=yes" -o
 "ControlPath=/home/fred/.ssh/controlmasters/%r@%h:%p" server.example.org
```

9 http://man.openbsd.org/ssh.1

Then subsequent sessions are connected to the control master via socket at the end of the control path.

```
$ ssh -o "ControlPath=/home/fred/.ssh/controlmasters/%r@%h:%p"
 server.example.org
```

And of course all that can be put into ssh_config(5)[10] as shown in the previous section. Starting with 6.7, the combination of %r@%h:%p and variations on it can be replaced with %C which by itself generates a SHA1 hash from the concatenation of %l%h%p%r.

```
$ ssh -S ~/.ssh/controlmasters/%C server.example.org
```

There are two advantages with that. One is that the hash can be shorter than its combined elements while still uniquely identifying the connection. The other is that it obfuscates the connection information which would have been otherwise displayed in the socket's name.

22.2.1 Ending Multiplexed Connections

One way to end multiplexed sessions is to exit all related SSH sessions, including the control master. If the control master has been placed in the background using **ControlPersist**, then it will be necessary to stop it with **-O** and either 'stop' or 'exit'. That also requires knowing the full path to and filename of the control socket as used in the creation of the master session, if it has not been defined in a shortcut in ssh_config(5)[11].

```
$ ssh -O stop server1
$ ssh -O stop -S ~/.ssh/controlmasters/%C server1.example.org
```

The multiplex command **-O stop** will gracefully shutdown multiplexing. After issuing the command, the control socket is removed and no new multiplexed sessions are accepted for that master. Existing connections are allowed to continue and the master connection will persist until the last multiplexed connection closes.

In contrast, the multiplex command **-O exit** removes the control socket and immediately terminates all existing connections.

Again, the directive **ControlPersist** can also be set to timeout after a set period of disuse. The time interval there is written in a time format listed in sshd_config(5)[12] or else defaults to seconds if no units are given. It will cause the master connection to automatically close if it has no client connections for the specified time.

```
Host server1
        HostName server1.example.org
        ControlPath ~/.ssh/controlmasters/%C
        ControlMaster yes
        ControlPersist 2h
```

10 http://man.openbsd.org/ssh_config.5
11 http://man.openbsd.org/ssh_config.5
12 http://man.openbsd.org/sshd_config.5

The above example lets the control master timeout after 2 hours of inactivity. Care should be used with persistent control sockets. A user that can read and write to a control socket can establish new connections without further authentication.

22.2.2 Multiplexing Options

The values for configuring session multiplexing can be set either in the user-specific ssh_config(5)[13], or the global **/etc/ssh/ssh_config**, or using parameters when running from the shell or a script. **ControlMaster** can be overridden in the run time arguments to re-use an existing master when **ControlMaster** is set to 'yes', by setting it explicitly to 'no':

```
$ ssh -o "ControlMaster=no" server.example.org
```

ControlMaster accepts five different values: 'no', 'yes', 'ask', 'auto', and 'autoask'.

- 'no' is the default. New sessions will not try to connect to an established master session, but additional sessions can still multiplex by connecting explicitly to an existing socket.
- 'yes' creates a new master session each time, unless explicitly overridden. The new master session will listen for connections.
- 'ask' creates a new master each time, unless overridden, which listen for connections. If overridden, ssh-askpass(1)[14] will popup an message in X to ask the master session owner to approve or deny the request. If the request is denied, then the session being created falls back to being a regular, standalone session.
- 'auto' creates a master session automatically but if there is a master session already available, subsequent sessions are automatically multiplexed.
- 'autoask' automatically assumes that if a master session exists, that subsequent sessions should be multiplexed, but ask first before adding a session.

Refused connections are logged to the master session.

```
Host *
        ControlMaster ask
```

ControlPath can be a fixed string or include any of several pre-defined variables described in the TOKENS section of the ssh_config(5)[15]. **%L** is for the first component of the local host name and **%l** for the full local host name. **%h** is the target host name and **%n** is the original target host name and **%p** is for the destination port on the remote server. **%r** is for the remote user name and **%u** for the user running ssh(1)[16]. Or they can be combined as **%C** which is a SHA1 hash produced from **%l%h%p%r**.

```
Host *
        ControlMaster ask
        ControlPath ~/.ssh/controlmasters/%C
```

13 http://man.openbsd.org/ssh_config.5
14 http://man.openbsd.org/ssh-askpass.1
15 http://man.openbsd.org/ssh_config.5#TOKENS
16 http://man.openbsd.org/ssh.1

ControlPersist accepts 'yes', 'no' or a time interval. If a time interval is given, the default is in seconds. Units can extend the time to minutes, hours, days, weeks or a combination. If 'yes' the master connection stays in the background indefinitely.

```
Host *
        ControlMaster ask
        ControlPath ~/.ssh/controlmasters/%C
        ControlPersist 10m
```

Port Forwarding After the Fact

It is possible to request port forwarding without having to establish new connections. Here we forward port 8080 on the local host to port 80 on the remote host using **-L**:

```
$ ssh -O forward -L 8080:localhost:80 -S
 ~/.ssh/controlmasters/fred@server.example.org:22  server.example.org
```

The same can be done for remote forwarding, using **-R**. The escape sequence **~C** is not available for multiplexed sessions, so **-O forward** is the only way to add port forwarding on the fly.

Port forwarding can be canceled without having to close any sessions using **-O cancel**.

```
$ ssh -O cancel -L 8080:localhost:80 -S
 ~/.ssh/controlmasters/fred@server.example.org:22  server.example.org
```

The exact same syntax used for forwarding is used for cancelling. However, there is no way currently to look up which ports are being forwarded and how they are forwarded.

22.3 Additional Notes About Multiplexing

Never use any publicly accessible directory for the control path sockets. Place those sockets in a directory somewhere else, one to which only your account has access. For example **~/.ssh/socket/** would be a much safer choice and **/tmp/** would be a bad choice.

In cases where it is useful to have a standing connection, it is always possible to combine multiplexing with other options, such as **-f** or **-N** to have the control master drop to the background upon connection and not load a shell.

In sshd_config(5)[17], the directive **MaxSessions** specifies the maximum number of open sessions permitted per network connection. This is used when multiplexing ssh sessions over a single TCP connection. Setting **MaxSessions** to 1 disables multiplexing and setting it to 0 disables login/shell/subsystem sessions completely. The default is 10. The **MaxSessions** directive can also be set under a **Match** conditional block so that different settings are available for different conditions.

17 http://man.openbsd.org/sshd_config.5

22.3.1 Errors Preventing Multiplexing

The directory used for holding the control sockets must be on a file system that actually allows the creation of sockets. AFS is one example that doesn't, as are some implementations of HFS+. If an attempt is made at creating a socket on a file system that does not allow it, an error something like the following will occur:

```
$ ssh -M -S /home/fred/.ssh/mux 192.0.2.57
muxserver_listen: link mux listener /home/fred/.ssh/mux.vjfeIFFzHnhgHoOV =>
 /home/fred/.ssh/mux: Operation not permitted
```

A similar issue was seen when trying to make Unix domain sockets on OverlayFS filesystems, which are often used with Docker, prior to Linux 4.7 kernel[61].

If the file system cannot be reconfigured to allow sockets, then the only other option is to place the control path socket somewhere else on a file system that does support creation of sockets.

22.3.2 Observing Multiplexing

It is possible to make some rough measurements to show the differences between multiplexed connections and one-off connections as shown in the tables and figures above.

Measuring the Number of TCP Connections

Tables 1 and 2 above use output from netstat(8)[18] and awk(1)[19] to show the number of TCP connections corresponding to the number of SSH connections.

```
#!/bin/sh
netstat -nt | awk 'NR == 2'

ssh -f server.example.org sleep 60
echo # one connection
netstat -nt | awk '$5 ~ /:22$/'

ssh -f server.example.org sleep 60
echo # two connections
netstat -nt | awk '$5 ~ /:22$/'

ssh -f server.example.org sleep 60
echo # three connections
netstat -nt | awk '$5 ~ /:22$/'

echo Table 1
```

And

```
#!/bin/sh
netstat -nt | awk 'NR == 2'

ssh -f -M -
S ~/.ssh/demo server.example.org sleep 60
echo # one connection
netstat -nt | awk '$5 ~ /:22$/'

ssh -f -
S ~/.ssh/demo server.example.org sleep 60 &
echo # two connections
netstat -nt | awk '$5 ~ /:22$/'

ssh -f -
S ~/.ssh/demo server.example.org sleep 60 &
echo # three connections
netstat -nt | awk '$5 ~ /:22$/'

echo Table 2
```

18 http://linux.die.net/man/8/netstat
19 http://linux.die.net/man/1/awk

The sleep(1)[20] period can be increased if more delay is needed. While the connections are active, it is also possible to look on the server using ps(1)[21], such as with `ps uwx`, to see the processes as a means of verifying that multiple connections are indeed being made.

Measuring Response Time

Measuring the response time requires setting up keys with an agent first so that the agent handles the authentication and eliminates authentication as a source of delay. See the section on keys, if needed. All of the response time tests below will depend on using keys for authentication.

For a one-off connection, just add time(1)[22] to check how long access takes.

```
$ time ssh -i ~/.ssh/rsakey server.example.org true
```

For a multiplexed connection, a master control session must be established. Then subsequent connections will show an increase in speed.

```
$ ssh -f -M -S ~/.ssh/demo -i ~/.ssh/rsakey server.example.org
$ time ssh  -S ~/.ssh/demo -i ~/.ssh/rsakey server.example.org true
$ time ssh  -S ~/.ssh/demo -i ~/.ssh/rsakey server.example.org true
```

These response times are approximate but the difference is nonetheless large enough to be seen between one-off connections and multiplexed connections.

Keeping sessions open

Probably you also want to keep your sessions open when inactive. You can configure it using **ServerAliveInterval** and **ServerAliveCountMax** options, see ssh_config(5)[23] for the details.

22.4 Multiplexing HTTPS and SSH Using sslh

A different kind of multiplexing is when more than one protocol is carried over the same port. sslh[24] does just that with SSL and SSH. It figures out what kind of connection is coming in and forwards it to the appropriate service. Thus it allows a server to receive both HTTPS and SSH over the same port, making it possible to connect to the server from behind restrictive firewalls in some cases. It does not hide SSH. Even a quick scan for SSH on the listening port, say with scanssh(1)[25], will show that it is there. Please note that this method only works with simpler packet filters, such as PF or nftables(8) and its front-ends, firewalld and UFW, which filter based on the destination port and won't fool

20 http://linux.die.net/man/1/sleep
21 http://linux.die.net/man/1/ps
22 http://linux.die.net/man/1/time
23 http://man.openbsd.org/ssh_config.5
24 https://www.rutschle.net/tech/sslh/README.html
25 https://en.wikibooks.org/wiki/OpenSSH/Third_Party_Utilities#scanssh

protocol analysis and application layer filters, such as Zorp, which filter based on the actual protocol used. Here's how sslh(8)[26] can be installed in four steps:

- First install your web server and configure it to accept HTTPS requests. Be sure that it listens for HTTPS on the localhost only. It can help in that regard if it is on a non-standard port, say 2443, instead of 443.
- Next, set your SSH server to accept connections on the localhost for port 22. It really could be any port, but port 22 is the standard for SSH.
- Next, create an unprivileged user to run sslh(8)[27]. The example below has the user 'sslh' for the unprivileged account.
- Lastly, install and launch sslh(8)[28] so that it listens on port 443 and forwards HTTPS and SSH to the appropriate ports on the local host. Substitute the external IP number for your machine. The actual name of the executable and path may vary from system to system:

```
$ /usr/local/sbin/sslh-fork -u sslh -p xx.yy.zz.aa:443 --tls 127.0.0.1:2443
 --ssh 127.0.0.1:22
```

Another option is to use a configuration file with sslh(8)[29] and not pass any parameters at runtime. There should be at least one sample configuration file, **basic.cfg** or **example.cfg** , included in the package when it is installed. The finished configuration file should look something like this:

```
user: "sslh";
listen: ( { host: "xx.yy.zz.aa"; port: "443" } );
on-timeout: "ssl";
protocols:
(
   { name: "ssh"; host: "localhost"; port: "22"; probe: "builtin"; },
   { name: "ssl"; host: "localhost"; port: "2443"; probe: "builtin"; }
);
```

Mind the quotes, commas, and semicolons.

If an old version of SSH is used in conjunction with now-deprecated TCP Wrappers, as described in hosts_access(5)[30], then the option **service:** works to provide the name of the service that they need.

```
    { name: "ssh"; service: "ssh"; host: "localhost"; port: "22"; probe:
 "builtin"; },
```

If TCP Wrappers are not used, which is most likely the case, then **service:** is not needed.

Runtime parameters override any configuration file settings that may be in place. sslh(8)[31] supports the protocols HTTP, SSL, SSH, OpenVPN, tinc, and XMPP out of the box. But actually any protocol that can be identified by regular expression pattern matching can be

26 http://linux.die.net/man/8/sslh
27 http://linux.die.net/man/8/sslh
28 http://linux.die.net/man/8/sslh
29 http://linux.die.net/man/8/sslh
30 http://linux.die.net/man/5/hosts_access
31 http://linux.die.net/man/8/sslh

used. There are two variants of sslh[32], a forked version (sslh or sslh-fork) and a single-threaded version (sslh-select). See the project web site for more details: [33]

32 https://www.rutschle.net/tech/sslh/README.html
33 https://www.rutschle.net/tech/sslh/README.html

23 Proxies and Jump Hosts

A proxy is an intermediary that forwards requests from clients to other servers. Performance improvement, load balancing, security or access control are some reasons proxies are used.

23.1 Jump Hosts – Passing Through a Gateway or Two

It is possible to connect to another host via one or more intermediaries so that the client can act as if the connection were direct.

The main method is to use an SSH connection to forward the SSH protocol through one or more *jump* hosts, using the **ProxyJump** directive, to an SSH server running on the target destination host. This is the most secure method because encryption is end-to-end. In addition to whatever other encryption goes on, the end points of the chain encrypt and decrypt each other's traffic. So the traffic passing through the intermediate hosts is always encrypted. But this method cannot be used if the intermediate hosts deny port forwarding.

Using the **ProxyCommand** option to invoke Netcat as the last in the chain is a variation of this for very old clients. The SSH protocol is forwarded by `nc` instead of `ssh`. Attention must also be paid to whether or not the username changes from host to host in the chain of SSH connections. The outdated netcat[1] method does not allow a change of username. Other methods do.

When port forwarding is available the easiest way is to use **ProxyJump** in the configuration file or **-J** as a run-time parameter. An example of **-J** usage is:

```
$ ssh -J firewall.example.org:22 server2.example.org
```

The **ProxyJump** directive (**-J**) is so useful it has an entire sub-section below[2].

In older versions **-J** is not available. In this case the safest and most straightforward way is to use ssh(1)[3]'s stdio forwarding (**-W**) mode to "bounce" the connection through an intermediate host.

```
$ ssh -o ProxyCommand="ssh -W %h:%p firewall.example.org" server2.example.org
```

This approach supports port-forwarding without further tricks.

1 http://man.openbsd.org/nc.1
2 #Passing_Through_One_or_More_Gateways_Using_ProxyJump
3 http://man.openbsd.org/ssh.1

> A note about using **%h**, **%u**, and other expansion tokens from dodgy remote systems: In older versions of OpenSSH, an invalid user name or host name could be used to pass shell metacharacters via the expansion tokens. The problem is described in CVE-2023-51385[4] and mitigated somewhat in OpenSSH 9.6[5] and later. It refers to untrusted proxy strings, such as may come from unreliable version control services and elsewhere. The mitigation is partial as it is infeasible to filter all the shell metacharacters which may be relevant to user-supplied commands.

Even older clients don't support the **-W** option. In this case `ssh -tt` may be used. This forces TTY allocation and passes the SSH traffic as though typed, though this is less secure. To connect to *server2* via *firewall* as the jump host:

```
$ ssh -tt firewall.example.com ssh -tt server2.example.org
```

That example opens an SSH session to the remote machine. You can also pass commands. For example, to reattach to a remote screen session using screen[6] you can do the following:

```
$ ssh -tt firewall.example.com ssh -tt server2.example.org screen -dR
```

The chain can be arbitrarily long and is not limited to just two hosts. The disadvantage of this approach over stdio-forwarding at the network layer with **-W** or **-J** is that your session, any forwarded agent, X11 server, and sockets are exposed to the intermediate hosts.

23.1.1 Passing Through One or More Gateways Using ProxyJump

Starting from OpenSSH 7.3, released August 2016[62], the easiest way to pass through one or more jump hosts is with the **ProxyJump** directive in ssh_config(5)[7].

```
Host server2
        HostName 192.168.5.38
        ProxyJump user1@jumphost1.example.org:22
        User fred
```

Multiple jump hosts can be specified as a comma-separated list. The hosts will be visited in the order listed.

```
Host server3
        HostName 192.168.5.38
        ProxyJump
 user1@jumphost1.example.org:22,user2@jumphost2.example.org:2222
        User fred
```

It also has the shortcut of **-J** when using it as a run-time parameter.

```
$ ssh -J user1@jumphost1.example.org:22 fred@192.168.5.38
```

Multiple jump hosts can be chained in the same way.

4 https://nvd.nist.gov/vuln/detail/CVE-2023-51385
5 https://www.openssh.com/txt/release-9.6
6 http://linux.die.net/man/1/screen
7 http://man.openbsd.org/ssh_config.5

```
$ ssh -J user1@jumphost1.example.org:22,user2@jumphost2.example.org:2222
 fred@192.168.5.38
```

It is not possible to use both the **ProxyJump** and **ProxyCommand** directives in the same host configuration. The first one found is used and then the other blocked.

23.1.2 Transiting a Jump Host Which Has Multiple RDomains / Routing Tables

When passing through a jump host which has its relevant interfaces each on a different rdomain(4)[8], it will be necessary to manipulate the routing tables manually. Specifically that means going back to an older method of transit which relies on netcat[9] and uses **ProxyCommand** instead of **ProxyJump** so that route(8)[10] can be added in the middle. Here is an example doing so using only run time parameters to pass through to rdomain 1:

```
$ ssh -o ProxyCommand="ssh user1@jumphost.example.org route -T 1 exec nc %h %p"
 fred@server.example.org
```

That configuration can be made persistent by adding it to the client configuration file, ssh_config(5)[11]:

```
Host jump jumphost.example.org
        HostName jumphost.example.org
        User user1
        IdentitiesOnly yes
        IdentityFile ~/.ssh/jump_key

Host server server.example.com
        HostName 10.100.200.44
        User fred
        IdentitiesOnly yes
        IdentityFile ~/.ssh/inside_key
        ProxyCommand ssh jump route -T 1 exec nc %h %p
```

With those settings, it is then possible to connect to the host on the LAN via the jump host simply with the line `ssh server` and all the settings will be taken into use.

Otherwise, the recommended way would have been to use **ProxyJump**. For more on using the older **ProxyCommand** directive see the section below, ProxyCommand with Netcat[12].

23.1.3 Conditional Use of Jump Hosts

It is possible to use **Match exec** to select for difficult patterns or otherwise make complex decisions, such as which network the client is connecting from or the network connectivity of the available network(s). Below are two cases for how to automatically choose when to use **ProxyJump** to connect via an intermediate host. The first example is for occasions when

8 http://man.openbsd.org/rdomain.4
9 http://man.openbsd.org/nc.1
10 http://man.openbsd.org/route.8
11 http://man.openbsd.org/ssh_config.5
12 https://en.wikibooks.org/wiki/OpenSSH/Cookbook/Proxies_and_Jump_Hosts#ProxyCommand_with_Netcat

there is no local IPv6 connectivity when connecting to a remote machine which has only IPv6[63] [64]. The second case[65] is for occasions when the target machines are on another network.

```
Match host ipv6only.example.org
        User fred

Match host ipv6only.example.org !exec "route -n get -inet6 %h"
        ProxyJump dualstack.example.org
```

With those settings, connections to the machine *ipv6only.example.org* will go via a jump host only if it is not directly accessible via IPv6.

On GNU/Linux systems that capability would be more complicated.

```
Match host ipv6only.example.org
        User fred

Match host ipv6only.example.org !exec "/sbin/ip route get $(host -t AAAA %h |
 sed 's/^.* //')"
        ProxyJump dualstack.example.org
```

The scripts are invoked using the shell named in the **$SHELL** environment variable. Remember that configuration directives are applied according to the first one to match. So specific rules go near the top and more general ones go at the end.

Another way is to use the nc(1)[13] utility to check for connectivity to the target host directly at the port in question.

```
Match !host jumphost.example.org !exec "nc -z -w 1 %h %p"
        ProxyJump jumphost.example.org
```

Since the above assumes that only one jump host is ever used, it might be combined with other **Match** criteria for more precision.

```
Match host 192.168.1.* !host jumphostone.example.org !exec "nc -z -w 1 %h %p"
        ProxyJump jumphostone.example.org

Match host 192.168.2.* !host jumphosttwo.example.org !exec "nc -z -w 1 %h %p"
        ProxyJump jumphosttwo.example.org
```

That would catch all connections going to the network 192.168.1.*, if there is no direct connection, and send them through the first jump host. Likewise it would also send all connections to the network 192.168.2.* through the second jump host if there is no direct connection. But if there is a direct connection the client will proceed without the jump host.

Of course the tests will have to be more complicated if the client machine moves between several different networks with the same numbering.

13 http://man.openbsd.org/nc.1

23.1.4 Using Canonical Host Names Which Are Behind Jump Hosts

Some local area networks (LANs) have their own internal domain name service to assign its own host names. These names are not accessible to systems outside the LAN. So therefore it is not possible to use the **-J** or **ProxyJump** option from the outside because the client would not be able to look up the names on the LAN on the other side of the jump host. However, the jump host itself can look these names up, so the **ProxyCommand** option can be used instead to call an SSH client on the jump host and use its capabilities to look up a name on the LAN.

In the following sequence, the absence of outside DNS records for the inner host is shown. Then a connection is made to the inner host via the jump host using the jump host's SSH client and the **-W** option. Keys or certificates are the recommended way of connecting, but passwords are used in this example so that the stages are more visible:

```
$ host fuloong04.localnet.lan
Host fuloong04.localnet.lan not found: 3(NXDOMAIN)

$ host fuloong04
Host fuloong04 not found: 2(SERVFAIL)

$ ssh -o ProxyCommand="ssh -W fuloong04.localnet.lan:22 jumphost.example.org"
 fred@fuloong04
fred@jumphost.example.org's password:
fred@fuloong04's password:
Last login: Sun May 24 04:06:21 2020 from 203.0.113.131
OpenBSD 6.7-current (GENERIC) #44: Sun May 24 04:06:31 MDT 2020

$ host fuloong04.localnet.lan
fuloong04.localnet.lan has address 10.10.10.193
```

The **-W** option ensures that the connection is forwarded over the secure channel and just passes through the jump host without being decrypted. The jump host must both be able to do the DNS look up for LAN names as well as have an SSH client available.

Here is what the initial connection looks like as the client collects the host key and asks about it:

```
$ ssh -o ProxyCommand="ssh -W fuloong04.localnet.lan:22 jumphost.example.org"
 fred@fuloong04
fred@jumphost.example.org's password:
The authenticity of host 'fuloong04 (<no hostip for proxy command>)' can't be
 established.
ECDSA key fingerprint is SHA256:TGH6fbXRswaP7iR1OBrJuhJ+c1JiEvvc2GJ2krqaJy4.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added 'fuloong04' (ECDSA) to the list of known hosts.
fred@fuloong04's password:
Last login: Thu May  7 21:06:18 2020 from 203.0.113.131
OpenBSD 6.7-current (GENERIC) #44: Sun May 24 04:06:21 MDT 2020
```

Notice that while a name must be given in the usual position it will only serve to identify which host keys are to be associated with the connection. After that initial connection, a host key is saved in **known_hosts** for the name given as destination *fuloong04* not the name given in the **ProxyCommand** option:

```
$ ssh-keygen -F fuloong04
# Host fuloong04 found: line 55
fuloong04 ecdsa-sha2-nistp256 AAAAE2VjZHNhLXNoYTItbmlzdHAyNTYAAAAIbmlzdHAyNTYAAA
```

```
BBBKwZOXY7
wP1lCvlv5YC64QvWPxSrhCaODn+pK4b97jjeUqGy/wII5zCnuv6WZKCEjSQnKFP+8K9cmSGnIvUisPg=

$ ssh-keygen -F fuloong04.localnet.lan
```

The target host name just serves to identify which host keys to expect. It has no association with the actual host name or address passed in the **ProxyCommand** option and could even be a random string. Above *fuloong04* was used by itself but, again, it could be any random string or even overridden by using the **HostKeyAlias** option.

As usual, these SSH options can be recorded in a permanent shortcut within ssh_config(5)[14] to reduce effort in typing and to avoid errors.

23.1.5 Old Methods of Passing Through Jump Hosts

A note on old methods: These old methods are deprecated because newer versions of OpenSSH provide easier ways to pass through a jump host. See instead the section Passing Through One or More Gateways Using ProxyJump[15] above. However, some long term support versions of various GNU/Linux distributions may keep very old versions of OpenSSH around on purpose. So while these are unlikely to be encountered, there is still a possibility to need these methods for some years to come. In old versions of OpenSSH, specifically version 7.2 and earlier, passing through one or more gateways is more complex and requires use of stdio forwarding or, prior to 5.4, use of the netcat(1)[16] utility.

Old: Passing Through a Gateway Using stdio Forwarding (Netcat Mode)

Between OpenSSH 5.4[66] and 7.2 inclusive, a 'netcat mode' can connect stdio on the client to a single port forwarded on the server. This can also be used to connect using ssh(1)[17], but it needs the **ProxyCommand** option either as a run time parameter or as part of **~/.ssh/config**. However, it no longer needs netcat(1)[18] to be installed on the intermediary machine(s). Here is an example of using it in a run time parameter.

```
$ ssh -o ProxyCommand="ssh -W %h:%p jumphost.example.org" server.example.org
```

In that example, authentication will happen twice, first on the jump host and then on the final host where it will bring up a shell.

The syntax is the same if the gateway is identified in the configuration file. ssh(1)[19] expands the full name of the gateway and the destination from the configuration file. The following allows the destination host to be reached by entering `ssh` *`server`* in the terminal.

14 http://man.openbsd.org/ssh_config.5
15 https://en.wikibooks.org/wiki/OpenSSH/Cookbook/Proxies_and_Jump_Hosts#Passing_Through_One_or_More_Gateways_Using_ProxyJump
16 http://man.openbsd.org/nc.1
17 http://man.openbsd.org/ssh.1
18 http://man.openbsd.org/nc.1
19 http://man.openbsd.org/ssh.1

```
Host server
        Hostname server.example.org
        ProxyCommand ssh -W %h:%p jumphost.example.org
```

The same can be done for SFTP. Here the destination SFTP server can be reached by entering `sftp` *`sftpserver`* and the configuration file takes care of the rest. If there is a mix up with the final host key, then it is necessary to add in **HostKeyAlias** to explicitly name which key will be used to identify the destination system.

```
Host sftpserver
        HostName sftpserver.example.org
        HostKeyAlias sftpserver.example.org
        ProxyCommand ssh -W %h:%p jumphost.example.org
```

It is possible to add the key for the gateway to the ssh-agent which you have running or else specify it in the configuration file. The option **User** refers to the user name on the destination. If the user is the same on both the destination and the originating machine, then it does not need to be used. If the user name is different on the gateway, then the **-l** option can be used in the **ProxyCommand** option. Here, the user 'fred' on the local machine, logs into the gateway as 'fred2' and into the destination server as 'fred3'.

```
Host server
        HostName server.example.org
        User fred3
        ProxyCommand ssh -l fred2 -i /home/fred/.ssh/rsa_key -W %h:%p
 jumphost.example.org
```

If both the gateway and destination are using keys, then the option **IdentityFile** in the config is used to point to the gateway's private key and the option **IdentityFile** specified on the commandline points at the destination's private key.

```
Host jump
        HostName server.example.org
        IdentityFile /home/fred/.ssh/rsa_key_2
        ProxyCommand ssh -i /home/fred/.ssh/rsa_key -W %h:%p
 jumphost.example.org
```

The old way prior to OpenSSH 5.4 used netcat, **nc(1)**.

```
Host server
        Hostname server.example.org
        ProxyCommand ssh jumphost.example.org nc %h %p
```

But that should not be used anymore and the netcat mode, provided by **-W**, should be used instead. The new way does not require netcat[20] at all on any of the machines.

Old: Recursively Chaining Gateways Using stdio Forwarding

The easy way to do this is with **ProxyJump** which is available starting with OpenSSH 7.3 mentioned above. For older versions, if the route always has the same hosts in the same

20 `http://man.openbsd.org/nc.1`

order, then a straightforward chain can be put in the configuration file. Here three hosts are chained with the destination being given the shortcut *machine3*.

```
Host machine1
        Hostname server.example.org
        User fred
        IdentityFile /home/fred/.ssh/machine1_e25519
        Port 2222

Host machine2
        Hostname 192.168.15.21
        User fred
        IdentityFile /home/fred/.ssh/machine2_e25519
        Port 2222
        ProxyCommand ssh -W %h:%p machine1

Host machine3
        Hostname 10.42.0.144
        User fred
        IdentityFile /home/fred/.ssh/machine3_e25519
        Port 2222
        ProxyCommand ssh -W %h:%p machine2
```

Thus any machine in the chain can be reached with a single line. For example, the final machine can be reached with `ssh` *`machine3`* and worked with as normal. This includes port forwarding and any other capabilities.

Only **hostname** and, for second and subsequent hosts, **ProxyCommand** are needed for each **Host** directive. If keys are not used, then **IdentityFile** is not needed. If the user is the same for all hosts, the that can be skipped. And if the port is the default, then **Port** can be skipped. If using many keys in the agent at the same time and the error "too many authentication" pops up on the client end, it might be necessary to add **IdentitiesOnly yes** to each host's configuration.

Old: Recursively Chaining an Arbitrary Number of Hosts

Again, the easy way to do this is with **ProxyJump** should be used when available, which is starting with OpenSSH 7.3. For older versions, it is possible to make the configuration more abstract and allow passing through an arbitrary number of gateways. You can set the user name with **-l** thanks to the token, but that user name will be used for all host that you connect to or through.

```
Host */*
        ProxyCommand ssh %r@$(dirname %h) -W $(basename %h):%p
```

In this way hosts are separated with a slash (/) and can be arbitrary in number[67].

```
$ ssh host1/host2/host3/host4
```

There are limitations resulting from using the slash as a separator, as there would be with other symbols. However, it allows use of dirname(1)[21] and basename(1)[22] to process the host names.

21 http://man.openbsd.org/dirname.1
22 http://man.openbsd.org/basename.1

The following configuration uses sed(1)[23] to allow different port numbers and user names using the plus sign (+) as the delimiter for hosts, a colon (:) for ports, and an percentage sign (%) for user names. The basic structure is ***ssh -W $() $()*** and where **%h** is substituted for the target host name.

```
Host *+*
        ProxyCommand ssh -W $(echo %h | sed 's/^.*+//;s/^\([^:]*$\)/\1:22/')
 $(echo %h | sed 's/+[^+]*$//;s/\([^+%%]*\)%%\([^+]*\)$/\2 -l \1/;s/:\([^:+]*\)$/
 -p \1/')
```

The port can be left off for the default of 22 or delimited with a colon (:) for non-standard values[68].

```
$ ssh host1+host2:2022+host3:2224
```

As-is, the colons confound sftp(1)[24], so the above configuration will only work with it using standard ports. If sftp(1)[25] is needed on non-standard ports then another delimiter, such as an underscore (_), can be configured.

Any user name except the final one can be specified for a given host using the designated delimiter, in the above it is a percentage sign (%). The destination host's user name is specified with **-l** and all others can be joined to their corresponding host name with the delimiter.

```
$ ssh -l user3 user1%host1+user2%host2+host3
```

If user names are specified, depending on the delimiter, ssh(1)[26] can be unable to match the final host to an IP number and the key fingerprint in *known_hosts*. In such cases, it will ask for verification each time the connection is established, but this should not be a problem if either the equal sign (=) or percentage sign (%) is used.

If keys are to be used then load them into an agent, then the client figures them out automatically if agent forwarding with the **-A** option is used. However, agent forwarding is not needed if the **ProxyJump** option (**-J**) is available. It is considered by many to actually be a security flaw and a general misfeature. So use the **ProxyJump** option instead if it is available. Also, because the default for **MaxAuthTries** on the server is 6, using keys normally in an agent will limit the number of keys or hops to 6, with server-side logging getting triggered after half that.

Old: ProxyCommand with Netcat

Another way, which was needed for OpenSSH 5.3 and older, is to use the **ProxyCommand** configuration directive and netcat[27]. The utility nc(1)[28] is for reading and writing network connections directly. It can be used to pass connections onward to a second machine. In this case, *login* is the final destination reached via the intermediary *jumphost*.

23 http://man.openbsd.org/sed.1
24 http://man.openbsd.org/sftp.1
25 http://man.openbsd.org/sftp.1
26 http://man.openbsd.org/ssh.1
27 http://man.openbsd.org/nc.1
28 http://man.openbsd.org/nc.1

```
$ ssh -o 'ProxyCommand ssh %h nc login.example.edu 22' \
      -o 'HostKeyAlias=login.example.edu' \
      jumphost.example.org
```

Keys and different login names can also be used. Using **ProxyCommand**, ssh(1)[29] will first connect to *jumphost* and then from there to login.example.edu. The **HostKeyAlias** directive is needed to look up the right key for login.example.edu, without it the key for *jumphost* will be tried and that will, of course, fail unless both have the same keys. The account *user2* exists on *jumphost*.

```
$ ssh -o 'ProxyCommand ssh -i key-rsa -l user2 %h nc login.example.edu 22' \
      -o 'HostKeyAlias=login.example.edu' \
      jumphost.example.org
```

It's also possible to make this arrangement more permanent and reduce typing by editing **ssh_config** Here a connection is made to *host2* via *host1*:

```
Host host2.example.org
        ProxyCommand ssh fred@host1.example.org nc %h %p
```

Here a connection is made to *server2* via *server1* using the shortcut name 'jump'.

```
Host jump
        ProxyCommand ssh %h nc server2.example.org 22
        HostKeyAlias server2.example.org
        Hostname server1.example.org
        User fred
```

It can be made more general:

```
Host gateway.example.org
        ProxyCommand none

Host *.example.org my-private-host
        ProxyCommand ssh myuser@gateway.example.org nc %h %p
```

The same can be done with sftp(1)[30] by passing parameters on to ssh(1)[31]. Here is a simple example with sftp(1)[32] where *machine1* is the jump host to connect to *machine2*. The user name is the same for both hosts.

```
$ sftp -o 'ProxyCommand=ssh %h nc machine2.example.edu 22' \
       -o 'HostKeyAlias=machine2.example.edu' \
       fred@machine1.example.edu
```

Here is a more complex example using a key for *server1* but regular password-based login for the SFTP server.

29 http://man.openbsd.org/ssh.1
30 http://man.openbsd.org/sftp.1
31 http://man.openbsd.org/ssh.1
32 http://man.openbsd.org/sftp.1

```
$ sftp -o 'ProxyCommand ssh -i /Volumes/Home/fred/.ssh/server1_rsa \
       -l user2 server1.example.edu nc sftp.example.edu 22'   \
       -o 'HostKeyAlias=sftp.example.edu' sftp.example.edu
```

If the user accounts names are different on the two machines, that works, too. Here, 'user2' is the account on the second machine which is the final target. The user 'fred' is the account on the intermediary or jump host.

```
$ ssh -l user2 \
      -o 'ProxyCommand ssh -l fred %h nc machine2.example.org 22' \
      -o 'HostKeyAlias machine2.example.org' \
      machine1.example.org
```

Old: ProxyCommand without Netcat, Using Bash's */dev/tcp/* Pseudo-device

On GNU/Linux jump hosts missing even nc(1)[33], but which have the Bash shell interpreter, the pseudo device /**dev**/**tcp**[69] can be another option. This makes use of some built-in Bash functionality, along with the cat(1)[34] utility:

```
$ ssh -o HostKeyAlias=server.example.org \
    -o ProxyCommand="ssh -l fred %h 'exec 3<>/dev/tcp/server.example.com/22; cat
 <&3 & cat >&3;kill $!'" fred@jumphost.example.com
```

It is necessary to set the **HostKeyAlias** to the host name or IP address of the destination because the SSH connection is just passing through the jump host and needs to expect the right key.

The main prerequisite for this method is having a login shell on a GNU/Linux jump host with Bash as the default shell. If another shell is used instead, such as the POSIX shell, then the pseudo device will not exist and an error will occur instead:

```
sh: 1: cannot create /dev/tcp/server.example.com/22: Directory nonexistent
```

With ksh(1)[35], a similar error will occur.

```
ksh: cannot create /dev/tcp/server.example.com/22: No such file or directory
```

The zsh(1)[36] shell will also be missing the /**dev**/**tcp** pseudo device but produce a different error.

```
zsh:1: no such file or directory: /dev/tcp/server.example.com/22
zsh:1: 3: bad file descriptor
zsh:1: 3: bad file descriptor
zsh:kill:1: not enough arguments
```

33 https://man.openbsd.org/nc.1
34 https://man.openbsd.org/cat.1
35 https://man.openbsd.org/ksh.1
36 https://linux.die.net/man/1/zsh

So, again, this is a process specific to the bash(1)[37] shell. Furthermore, this method is specific to Bash on GNU/Linux distros and will not be available with any of the BSDs or Mac OS as a jump host even if bash(1)[38] is present.

The full details of the connection process from the client perspective can be captured to a log file using the '*-E* option while increasing the verbosity of the debugging information.

```
$ ssh -vvv -E /tmp/ssh.dev.tcp.log \
    -o HostKeyAlias=server.example.org \
    -o ProxyCommand="ssh -l fred %h 'exec 3<>/dev/tcp/server.example.com/22; cat
 <&3 & cat >&3;kill $!'" fred@jumphost.example.com
```

This pseudo device method is included here mostly as a curiosity but can serve as a last resort in some edge cases. As mentioned before, all recent deployments of OpenSSH will have the **-J** option for **ProxyJump** instead.

23.2 Port Forwarding Through One or More Intermediate Hosts

Tunneling, also called port forwarding, is when a port on one machine mapped to a connection to a port on another machine. In that way remote services can be accessed as if they were local. Or in the case of reverse port forwarding, vice versa. Forwarding can be done directly from one machine to another or via a machine in the middle.

When available, the **ProxyJump** option is preferable. It works just as easily with port forwarding. Below a tunnel is set up from the *localhost* to *machine2* which is behind a jump host.

```
$ ssh -L 8900:localhost:80 -J jumphost.example.org machine2
```

Thus port 8900 on the *localhost* will actually be a tunnel to port 80 on *machine2*. It can be as simple as that. As with the usual **ProxyJump** option, jump hosts can be chained by joining them with commas.

```
$ ssh -L 8900:localhost:80 -J jumphost1.example.org,jumphost2.localnet.lan
 machine2
```

Alternative accounts and ports can be specified that way, too, if needed. This method works best with key- or certificate-based authentication. It is possible to use all the options in this way, such as **-X** for X11 forwarding. The same with **-D** for making a SOCKS proxy.

23.2.1 Old: Port Forwarding Via a Single Intermediate Host Without ProxyJump

Below we are setting up a tunnel from the *localhost* to *machine2*, which is behind a firewall, *machine1*. The tunnel will be via *machine1* which is publicly accessible and also has access to *machine2*.

37 https://linux.die.net/man/1/bash
38 https://linux.die.net/man/1/bash

```
$ ssh -L 2222:machine2.example.org:22 machine1.example.org
```

Next connecting to the tunnel will actually connect to the second host, *machine2*.

```
$ ssh -p 2222 remoteuser@localhost
```

Here is an example of running rsync(1)[39] between the two hosts using *machine1* as an intermediary with the above setup.

```
$ rsync -av -e "ssh -p 2222"  /path/to/some/dir/   localhost:/path/to/some/dir/
```

23.3 SOCKS Proxy

It is possible to connect via an intermediate machine using a SOCKS proxy. SOCKS4 and SOCKS5 proxies are currently supported by OpenSSH. SOCKS5[70] allows transparent traversal of a firewall or other barrier by an application and can use strong authentication with help of GSS-API. Dynamic application-level port forwarding allows the outgoing port to be allocated on the fly thus creating a proxy at the TCP session level.

Here the web browser can connect to the SOCKS proxy on port 3555 on the local host:

```
$ ssh -D 3555 server.example.org
```

Using ssh(1)[40] as a SOCKS5 proxy, or in any other capacity where forwarding is used, you can specify multiple ports in one action:

```
$ ssh -D 80 -D 8080 -f -C -q -N fred@server.example.org
```

You'll also want the DNS requests to go via your proxy. So, for example, in recent versions of Firefox, there may be an option ”Proxy DNS when using SOCKS v5” to check. Or in older versions, **about:config**needs **network.proxy.socks_remote_dns** set to **true** instead. However, in Chromium[71], you'll need to launch the browser while adding two run-time options –**proxy-server** and –**host-resolver-rules** to point to the proxy and tell the browser not to send any DNS requests over the open network.

It will be similar for other programs that support SOCKS proxies. So, you can tunnel Samba over ssh(1)[41], too.

Going via a jump host is just a matter of using the **ProxyJump** option:

```
$ ssh -D 8899 -J jumphost.example.org machine2
```

Chaining multiple jump hosts can be done this way too. See the earlier section on that for discussion and examples.

39 http://linux.die.net/man/1/rsync
40 http://man.openbsd.org/ssh.1
41 http://man.openbsd.org/ssh.1

23.3.1 Old: SOCKS Proxy Via a Single Intermediate Host

If you want to open a SOCKS proxy via an intermediate host, it is possible:

```
$ ssh -D 3555 -J user1@middle.example.org user2@server.example.org
```

On older clients, an extra step is needed.

```
$ ssh -L 8001:localhost:8002 user1@middle.example.org -t ssh -D 8002
 user2@server.example.org
```

In the second example, the client will see a SOCKS proxy on port 8001 on the local host, which is actually a connection to *machine1* and the traffic will ultimately enter and leave the net through *machine2*. Port 8001 on the local host connects to port 8002 on *machine1* which is a SOCKS proxy to *machine2*. The port numbers can be chosen to be whatever is needed, but forwarding privileged ports still requires root privileges.

23.4 SSH Over Tor

There are two ways to use SSH over Tor. One is using the client over Tor, the other is to host the server as an Onion service. Running the client over Tor allows its origin to be hidden. Hosting the server behind Tor allows its location to be hidden. The two methods can be combined.

23.4.1 Tunneling the SSH Client Over Tor with Netcat

> These instructions use the port number 9050 which corresponds to the Tor system daemon. For the Tor Browser, substitute with port 9150 and leave the Tor Browser open.

Besides using ssh(1)[42] as a SOCKS proxy, it is possible to tunnel the SSH protocol itself over another SOCKS proxy such as Tor[43]. It is anonymity software and a corresponding network that uses relay hosts to conceal a user's location and network activity. Its architecture is intended to prevent surveillance and traffic analysis. Tor can be used in cases where it is important to conceal the point of origin of the SSH client. It can also be used to connect to onion services. Unfortunately, connecting through Tor often comes at the expense of noticeable latency.

On the end point which the client sees, Tor is a regular SOCKS5 proxy and can be used like any other SOCKS5 proxy. So this is tunneling SSH over a SOCKS proxy. The easiest way to do this is with the `torsocks` utility if available, simply by preceding a SSH command with it:[72]

```
$ torsocks ssh fred@server.example.org
```

42 `http://man.openbsd.org/ssh.1`
43 `https://www.torproject.org/`

However, this can also be accomplished by using netcat[44]:

```
$ ssh -o ProxyCommand="nc -X 5 -x localhost:9050 %h %p" fred@server.example.org
```

When attempting a connection like this, it is very important that it does not leak information. In particular, the DNS lookup should occur over Tor and not be done by the client itself. Make sure that if **VerifyHostKeyDNS** is used that it be set to 'no'. The default is 'no' but check to be sure. It can be passed as a run-time argument to remove any doubt or uncertainty.

```
$ ssh -o VerifyHostKeyDNS=no -o ProxyCommand="nc -X 5 -x localhost:9050 %h %p"
 server.example.org
```

Using the netcat-openbsd nc(1)[45] package, this seems not to leak any DNS information. Other netcat packages might or might not be the same. It's also not clear if there are other ways in which this method might leak information. YMMV.

Since these methods proxy through Tor, all of them will also enable connecting to onion services. You can configure SSH to automatically connect to these through Tor, without affecting other types of connections. An entry similar to the following can be added to **ssh_config** or **~/.ssh/config**:

```
Host *.onion
        VerifyHostKeyDNS no
        ProxyCommand nc -x localhost:9050 -X 5 %h %p
```

You can further add **CanonicalizeHostname yes** before any **Host** declarations so that if you give onion services a nickname, SSH will apply this configuration after determining the hostname is a .onion address.

23.4.2 Providing SSH as an Onion Service

SSH can be served from an .onion address to provide anonymity and privacy for the client and, to a certain extent, the server. Neither will know where the other is located, though a lot of additional precautions must be taken to come close to anonymizing the server itself and at least some level of trusting the users will still be necessary.

The first step in making SSH available over Tor is to set up sshd_config(5)[46] so that the SSH service listens on the *localhost* address and only on the *localhost* address.

```
ListenAddress 127.0.0.1:22
```

Multiple **ListenAddress** directives can be used if multiple ports are to be made available. However, any **ListenAddress** directives provided should bind only to addresses in the 127.0.0.0/8 network or the IPv6 equivalent. Listening to any WAN or LAN address would defeat Tor's anonymity by allowing the SSH server to be identified from its public keys.

44 http://man.openbsd.org/nc.1
45 http://man.openbsd.org/nc.1
46 http://man.openbsd.org/sshd_config.5

The next step in serving SSH over Tor is to set up a Tor client with a hidden service forwarded to the local SSH server. Follow the instructions for installing a Tor client given at the Tor Project web site[73], but skip the part about the web server if it is not needed. Add the appropriate **HiddenServicePort** directive to match the address and that sshd(8)[47] is using.

```
HiddenServicePort 22 127.0.0.1:22
```

If necessary, add additional directives for additional ports. In newer versions of Tor, a 56-character version 3 onion address[74] can be made by adding `HiddenServiceVersion 3` to the Tor configuration in versions that support it.

Be sure that **HiddenServiceDir** points to a location appropriate for your system. The onion address of the new SSH service will then be in the file *hostname* inside the directory indicated by the **HiddenServiceDir** directive and will be accessible regardless of where it is on the net or how many layers of NAT have it buried. To use the SSH service and verify that it is actually available over Tor, see the preceding section on using the SSH client over Tor with Netcat[48].

Just making it available over Tor is not enough alone to anonymize the server. However, one of the other advantages of this method is NAT punching. If the SSH service is behind several layers of NAT, then providing it as an Onion service allows passing through those layers seamlessly without configuring each router at each layer. This eliminates the need for a reverse tunnel to an external server and works through an arbitrary number of layers of NAT such as are now found with mobile phone modems.

23.5 Passing Through a Gateway with an Ad Hoc VPN

Two subnets can be connected over SSH by configuring the network routing on the end points to use the tunnel. The result is a VPN. A drawback is that root access is needed on both hosts, or at least sudo(8)[49] access to ifconfig(8)[50] and route(8)[51]. A related more limited and more archaic approach, not presented here but which does not require root at the remote end, would be to use `ssh` to establish connectivity and then establish a network using PPP and `slirp`.[75]

Note that there are very few instances where use of a VPN is legitimately called for, not because VPNs are illegal (quite the opposite, indeed data protection laws in many countries make them absolutely compulsory to protect content in transit) but simply because OpenSSH is usually flexible enough to complete most routine sysadmin and operational tasks using normal SSH methods as and when required. This SSH ad-hoc VPN method is therefore needed only very rarely.

Take this example with two networks. One network has the address range 10.0.50.1 through 10.0.50.254. The other has the address range 172.16.99.1 through 172.16.99.254. Each has

47 http://man.openbsd.org/sshd.8
48 http://man.openbsd.org/OpenBSD-current/man1/nc.1
49 http://linux.die.net/man/8/sudo
50 http://man.openbsd.org/ifconfig.8
51 http://man.openbsd.org/route.8

a machine, 10.0.50.1 and 172.16.99.1 respectively, that will function as a gateway. Local machine numbering starts with 3 because 2 will be used for the tunnel interfaces on each LAN.

```
              +----10.0.50.1       172.16.99.1----+
              +    10.0.50.2 ===== 172.16.99.2    +
              |                                   |
10.0.50.3-----+                                   +---172.16.99.3
              |                                   |
10.0.50.4-----+                                   +---172.16.99.4
              |                                   |
10.0.50.5-----+                                   +---172.16.99.5
              |                                   |
10.0.50.etc---+                                   +---172.16.99.etc
              |                                   |
10.0.50.254---+                                   +---172.16.99.254
```

First a tun device is created on each machine, a virtual network device for point-to-point IP tunneling. Then the tun interfaces on these two gateways are then connected by an SSH tunnel. Each tun interface is assigned an IP address.

The tunnel connects machines 10.0.50.1 and 172.16.99.1 to each other, and each are already connected to their own local area network (LAN). Here is a VPN with the client as 10.0.50.0/24, remote as 172.16.99.0/24. First, set on the client:

```
$ ssh -f -w 0:1 192.0.2.15 true
$ ifconfig tun0 10.1.1.1 10.1.1.2 netmask 255.255.255.252
$ route add 172.16.99.0/24 10.1.1.2
```

On the server:

```
$ ifconfig tun1 10.1.1.2 10.1.1.1 netmask 255.255.255.252
$ route add 10.0.50.0/24 10.1.1.1
```

23.5.1 Troubleshooting an Ad Hoc VPN

There are several possible causes to the following kind of error message:

```
channel 0: open failed: connect failed: open failed
Tunnel forwarding failed
```

One possibility is that tunneling has not yet been enabled on the server. The SSH server's default configuration has tunneling turned off, so it must be enabled explicitly using the **PermitTunnel** configuration directive prior to attempting an ad hoc VPN. Failure to enable tunneling will result in an error like the above when connecting using the **-w** option in the client. The solution in that case for that is to set **PermitTunnel** to 'yes' on the server.

Another possibility is that either the remote system or the local system or both lack the necessary network interface pseudo-device. That can happen because either or both accounts lack the privileges necessary to create the tun(4)[52] devices on the fly. Several work-arounds exist, including using a privileged account to make the tun(4)[53] devices on each system in

52 https://man.openbsd.org/tun
53 https://man.openbsd.org/tun

advance. In other words, the solution is to manually create the necessary network interface pseudo-devices for the tunneling.

A third possibility is that the one or both of the accounts do not have proper permissions to access the network interface pseudo-devices. Check and, if necessary, correct group memberships.

References

1. "Statistics from the current scan results"[54]. OpenSSH.com. 2008.
2. "OpenSSH History"[55]. OpenSSH. Retrieved 2012-11-17.
3. "UNIX History Timeline"[56]. Éric Lévénez. Retrieved 2011-02-17.
4. "Hobbes' Internet Timeline"[57]. Robert H'obbes' Zakon. Retrieved 2011-02-17.
5. Howard Dahdah (2009). "The A-Z of Programming Languages: Bourne shell, or sh"[58]. Computerworld. Retrieved 2011-02-18.
6. Phil Zimmermann (1991). "Why I Wrote PGP"[59]. Massachusetts Institute of Technology. Retrieved 2011-02-18.
7. Bill Bryant; Theodor Ts'o (1988). "Designing an Authentication System: a Dialogue in Four Scenes"[60]. Retrieved 2011-02-17.
8. "Help:SSH 1.0.0 license"[61]. FUNET. Retrieved 2013-04-13.
9. Tatu Ylönen (1995-07-12). "ANNOUNCEMENT: Ssh (Secure Shell) Remote Login Program"[62]. [63]. Retrieved 2011-11-26. `{{cite web`[64]`}}`: External link in `|publisher=` (help[65])
10. "Help:SSH 1.2.12 license"[66]. friedl. Retrieved 2011-02-17.

54 http://www.openssh.com/usage/ssh-stats.html
55 http://openssh.com/history.html
56 http://www.levenez.com/unix/
57 http://www.zakon.org/robert/internet/timeline/
58 http://www.computerworld.com.au/article/279011/a-z_programming_languages_bourne_shell_sh/
59 http://www.mit.edu/~prz/EN/essays/WhyIWrotePGP.html
60 http://web.mit.edu/Kerberos/dialogue.html
61 ftp://ftp.funet.fi/pub/mirrors/ftp.cs.hut.fi/pub/ssh/old/ssh-1.0.0.tar.gz
62 https://groups.google.com/group/comp.security.unix/msg/67079d812a19f499?dmode=source&hl=en&pli=1
63 news://comp.security.unix
64 https://en.wikibooks.org/wiki/Template:Cite_web
65 https://en.wikibooks.org/wiki/Help:CS1_errors#param_has_ext_link
66 http://wwwcip.informatik.uni-erlangen.de/~msfriedl/LIC/ssh-1.2.12/COPYING

11.
"Help:SSH 1.2.12.92 license"[67]. friedl. Retrieved 2011-02-17.
12. [68]
13.
"OpenSSH Project History and Credits"[69]. OpenSSH. Retrieved 2011-03-10.
14.
Robert H'obbes' Zakon. "Hobbes' Internet Timeline"[70]. Zakon Group LLC. Retrieved 2011-02-17.
15.
Damien Miller (2013-11-29). "ChaCha20 and Poly1305 in OpenSSH"[71]. Retrieved 2014-04-26.
16. [72]
17. [73]
18.
"SSH 1.0.0 README"[74]. FUNET. 1995.
19.
"OpenSSH: Secure Shell integrated into OpenBSD operating system"[75]. LWN. 1999. Retrieved 2011-02-18.
20.
"European Parliament resolution on the existence of a global system for the interception of private and commercial communications (ECHELON interception system) (2001/2098(INI))"[76]. European Parliament. 2001. ECHELON A5-0264 2001. Retrieved 2011-02-18.
21.
"OpenSSH Manual Pages"[77]. OpenSSH. Retrieved 2011-02-17.
22.
"RFC 4251: The Secure Shell (SSH) Protocol Architecture"[78]. 2006. Retrieved 2013-10-31.
23.
"Why You Need To Stop Using FTP"[79]. JDPFu.com. 2011-07-10. Retrieved 2012-01-09.
24.
Manolis Tzanidakis (2011-09-09). "Stop Using FTP! How to Transfer Files Securely"[80]. Wazi. Retrieved 2012-01-09.

67 http://wwwcip.informatik.uni-erlangen.de/~msfriedl/LIC/ssh-1.2.12.92/COPYING
68 https://www.openssh.com/history.html
69 http://www.openssh.com/history.html
70 http://www.zakon.org/robert/internet/timeline/
71 http://blog.djm.net.au/2013/11/chacha20-and-poly1305-in-openssh.html
72 https://linux.die.net/man/5/sshd_config
73 https://lists.mindrot.org/pipermail/openssh-unix-announce/2015-March/000120.html
74 https://ftp.funet.fi/pub/mirrors/ftp.cs.hut.fi/pub/ssh/old/
75 http://lwn.net/1999/1028/a/openssh.html
76 http://www.europarl.europa.eu/sides/getDoc.do?type=TA&reference=P5-TA-2001-0441&format=XML&language=EN
77 http://www.openssh.com/manual.html
78 http://tools.ietf.org/html/rfc4251
79 http://blog.jdpfu.com/2011/07/10/why-you-need-to-stop-using-ftp
80 http://olex.openlogic.com/wazi/2011/stop-using-ftp-how-to-transfer-files-securely/

25.
Jay Ribak (2002). "Active FTP vs. Passive FTP, a Definitive Explanation"[81]. Slacksite.com. Retrieved 2020-03-20.
26.
Nils Provos (2003). "Privilege Separated OpenSSH"[82]. University of Michigan. Retrieved 2011-02-17.
27.
"OpenSSH 5.9 Release Notes"[83]. OpenSSH. 2011-09-06. Retrieved 2012-11-17.
28.
"OpenSSH 9.0 Release Notes"[84]. www.openssh.com. 2022-04-08. Retrieved 2022-04-10.
29.
"Service Name and Transport Protocol Port Number Registry"[85]. IETF. 2012.
30.
"OpenSSH 2.3.0p1 release notes"[86]. OpenSSH.com.
31.
Brian Hatch (2004). "SSH Host Key Protection"[87]. SecurityFocus. Retrieved 2013-04-14.
32.
Jake Edge (2008). "Debian vulnerability has widespread effects"[88]. LWN. Retrieved 2013-04-14.
33.
Stefan Tatschner (2020-10-15). "SSH (Reverse) Tunnel Through Websocket"[89]. Retrieved 2020-10-20.
34.
Niels Provos; Peter Honeyman (2001). "ScanSSH - Scanning the Internet for SSH Servers"[90] (PDF). Center for Information Technology Integration (CITI), University of Michigan. Retrieved 2016-03-05.
35.
"RFC 4252: The Secure Shell (SSH) Authentication Protocol"[91]. IETF. 2006. Retrieved 2021-08-10.
36. Tucker, Darren (2021-08-10). "Re: ssh authlog: Failed none for invalid user"[92]. OpenBSD. [93]. Retrieved 2021-08-10.

81 http://slacksite.com/other/ftp.html
82 http://www.citi.umich.edu/u/provos/ssh/privsep.html
83 http://www.openssh.com/txt/release-5.9
84 https://www.openssh.com/txt/release-9.0
85 http://www.ietf.org/assignments/service-names-port-numbers/service-names-port-numbers.txt
86 http://www.openssh.com/txt/release-2.3.0p1
87 http://www.securityfocus.com/infocus/1806
88 https://lwn.net/Articles/282230/
89 https://rumpelsepp.org/blog/ssh-through-websocket/
90 http://www.citi.umich.edu/u/provos/papers/scanssh.pdf
91 https://datatracker.ietf.org/doc/html/rfc4252#section-5.2
92 https://marc.info/?l=openbsd-misc&m=162858437916966&w=2
93 https://marc.info/?l=openbsd-misc&m=162858437916966&w=2

37. Tucker, Darren (2019-04-01). "IdentityFile vs IdentitiesOnly"[94]. *openssh-unix-dev mailing list.* [95]. Retrieved 2019-04-04.
38. "SSH Troubleshooting Guide"[96]. IT Tavern. 2023-01-17. Retrieved 2023-01-26.
39. "The OpenBSD Foundation 2016 Fundraising Campaign"[97]. The OpenBSD Foundation. 2016. Retrieved 2016-03-07.
40. "Sharing Terminal Sessions With Tmux And Screen"[98]. HowToForge. 2012.
41. "How Rsync Works"[99]. Samba.
42. "NEWS for rsync 2.6.0 (1 Jan 2004)"[100]. Samba. 2004-01-01. Retrieved 2020-05-02.
43. "openrsync imported into the tree"[101]. Undeadly. 2019-02-11. Retrieved 2020-05-10.
44. Dan Langille (2014-05-03). "zfs send on FreeBSD over ssh using mbuffer"[102]. Retrieved 2020-05-22.
45. "Openssh SFTP Chroot Code Execution"[103]. halfdog.net. 2018-01-07. Retrieved 2018-01-09.
46. "The Secure Shell (SSH) Authentication Protocol"[104]. IETF. 2006. Retrieved 2015-05-06.
47. Steve Friedl (2006-02-22). "An Illustrated Guide to SSH Agent Forwarding"[105]. Unixwiz.net. Retrieved 2013-04-27.
48. Daniel Robbins (2002-02-01). "Common threads: OpenSSH key management, Part 3"[106]. IBM. Retrieved 2013-04-27.

94 https://lists.mindrot.org/pipermail/openssh-unix-dev/2019-April/037700.html
95 https://lists.mindrot.org/pipermail/openssh-unix-dev/2019-April/037700.html
96 https://ittavern.com/ssh-troubleshooting-guide/
97 http://www.openbsdfoundation.org/campaign2016.html
98 http://www.howtoforge.com/sharing-terminal-sessions-with-tmux-and-screen
99 https://www.samba.org/rsync/how-rsync-works.html
100 https://download.samba.org/pub/rsync/src/rsync-2.6.0-NEWS
101 https://undeadly.org/cgi?action=article;sid=20190211081518
102 https://dan.langille.org/2014/05/03/zfs-send-on-freebsd-over-ssh-using-mbuffer/
103 https://www.halfdog.net/Security/2018/OpensshSftpChrootCodeExecution/
104 https://tools.ietf.org/html/rfc4252#section-7
105 http://www.unixwiz.net/techtips/ssh-agent-forwarding.html#chal
106 http://www.ibm.com/developerworks/library/l-keyc3/

49.
Vincent Bernat (2020-04-05). "Safer SSH agent forwarding"[107]. Retrieved 2020-10-04.
50.
"Managing multiple SSH agents"[108]. Wikimedia. Retrieved 2020-04-07.
51.
Damien Miller (2021-12-16). "SSH agent restriction"[109]. OpenSSH. Retrieved 2022-03-06.
52.
"OpenSSH U2F/FIDO support in base"[110]. OpenBSD-Tech Mailing List. 2019-11-14. Retrieved 2021-03-24.
53. Miller, Damien (2023-03-01). "Why does ssh-keyscan not use .ssh/config?"[111]. mindrot.org. [112]. Retrieved 2023-03-01.
54.
Damien Miller (2015-02-01). "Key rotation in OpenSSH 6.8+"[113]. DJM's Personal Weblog. Retrieved 2016-03-05.
55.
Damien Miller (2015-02-17). "Hostkey rotation, redux"[114]. DJM's Personal Weblog. Retrieved 2016-03-05.
56.
Stephen Harris (2016-10-30). "Using SSH certificates"[115]. Retrieved 2020-05-07.
57.
Maggie Danger (2014-08-07). "How to Harden SSH with Identities and Certificates"[116]. Magnus Achim Deininger. Retrieved 2020-05-07.
58.
Mike Malone (2019-09-11). "If you're not using SSH certificates you're doing SSH wrong"[117]. Smallstep Labs, Inc. Retrieved 2020-05-07.
59.
Peter N M Hansteen (2011). "Firewalling with PF"[118].
60.
"OpenSSH 3.9 Release Notes"[119]. OpenSSH. 2004-08-18. Retrieved 2018-12-16.
61. Szeredi, Miklos (2016-06-16). "[GIT PULL[120] overlayfs fixes for 4.7-rc3"]. [121]. Retrieved 2018-10-05.

107 https://vincent.bernat.ch/en/blog/2020-safer-ssh-agent-forwarding
108 https://wikitech.wikimedia.org/wiki/Managing_multiple_SSH_agents#Linux_solutions
109 https://www.openssh.com/agent-restrict.html
110 https://marc.info/?l=openbsd-tech&m=157376801917387&w=2
111 https://lists.mindrot.org/pipermail/openssh-unix-dev/2023-March/040605.html
112 https://lists.mindrot.org/pipermail/openssh-unix-dev/2023-March/040605.html
113 http://blog.djm.net.au/2015/02/key-rotation-in-openssh-68.html
114 http://blog.djm.net.au/2015/02/hostkey-rotation-redux.html
115 https://www.sweharris.org/post/2016-10-30-ssh-certs/
116 https://ef.gy/hardening-ssh
117 https://smallstep.com/blog/use-ssh-certificates/
118 http://home.nuug.no/~peter/pf/
119 https://www.openssh.com/txt/release-3.9
120 https://lkml.org/lkml/2016/6/16/87
121 https://lkml.org/lkml/2016/6/16/87

62.
"OpenSSH 7.3 Release Notes"[122]. OpenSSH. 2016-08-01. Retrieved 2016-08-01.
63.
Peter Hessler (2018-12-04). "phessler"[123]. Mastodon. Retrieved 2018-12-05.
64.
Mike Lee Williams (2017-07-13). "Tunneling an SSH connection only when necessary using Match"[124]. Retrieved 2019-09-05.
65.
Andrew Hewus Fresh (2019-08-25). "afresh1"[125]. Mastodon. Retrieved 2019-09-05.
66.
"OpenSSH 5.4 Release Notes"[126]. OpenSSH. 2010-04-07. Retrieved 2013-04-19.
67.
Josh Hoblitt (2011-09-03). "Recursively chaining SSH ProxyCommand"[127]. [Read This Fine Material] from Joshua Hoblitt. Retrieved 2014-02-14.
68.
Mike Hommey (2016-02-08). "SSH through jump hosts, revisited"[128]. glandium. Retrieved 2016-02-09.
69.
"All about ssh ProxyCommand"[129]. Stack Exchange. 2011.
70.
"SOCKS Protocol version 5"[130]. IETF. Retrieved 2011-02-17.
71.
"Configuring a SOCKS proxy server in Chrome"[131]. The Chromium Projects. Retrieved 2016-12-10.
72.
"SSH Over Tor"[132]. The Tor Project. 2012-08-28. Retrieved 2013-05-04.
73.
"Configuring Hidden Services for Tor"[133]. The Tor Project, Inc. Retrieved 2016-12-09.
74.
"Tor Rendezvous Specification - Version 3"[134]. The Tor Project, Inc. 2015-05-26. Retrieved 2018-12-14.
75.
Jeremy Impson (2008-09-16). "pppsshslirp: create a PPP session through

122 http://www.openssh.com/txt/release-7.3
123 https://bsd.network/@phessler/101182487005125208
124 https://mike.place/2017/ssh-match/
125 https://bsd.network/@AFresh1/102674835140957188
126 http://www.openssh.com/txt/release-5.4
127 https://joshua.hoblitt.com/rtfm/2011/09/recursively_chaining_ssh_proxycommand/
128 https://glandium.org/blog/?p=3631
129 https://unix.stackexchange.com/questions/19604/all-about-ssh-proxycommand
130 http://tools.ietf.org/html/rfc1928
131 https://www.chromium.org/developers/design-documents/network-stack/socks-proxy
132 https://trac.torproject.org/projects/tor/wiki/doc/TorifyHOWTO/ssh
133 https://www.torproject.org/docs/tor-hidden-service#two
134 https://gitweb.torproject.org/torspec.git/tree/rend-spec-v3.txt

SSH to a remote machine to which you don't have root"[135]. Retrieved 2016-12-10.

135 http://jdimpson.livejournal.com/3330.html

24 References

- This page was last edited on 5 September 2020, at 10:35.
- Text is available under the Creative Commons Attribution-ShareAlike License[1]; additional terms may apply. By using this site, you agree to the Terms of Use[2] and Privacy Policy.[3]

1 http://creativecommons.org/licenses/by-sa/4.0/
2 http://foundation.wikimedia.org/wiki/Special:MyLanguage/Policy:Terms_of_Use
3 http://foundation.wikimedia.org/wiki/Special:MyLanguage/Policy:Privacy_policy

25 Contributors

Edits	User
1	1234qwer1234qwer4[1]
1	94rain[2]
1	AManNamedEdwan[3]
2	Abie Small[4]
27	Adrignola[5]
1	ApathySyndicate[6]
1	BethNaught[7]
2	Black Walnut[8]
2	Bryant1410[9]
1	Calimo[10]
7	Cedar101[11]
1	ChemicalyImbalanced[12]
1	CinoI[13]
11	DannyS712[14]
5	DavidCary[15]
4	Delapouite[16]
1	Djmdjm[17]
6	Djr13[18]
1	Erikweathers[19]
1	Gnatmat[20]

1 https://en.wikibooks.org/wiki/User:1234qwer1234qwer4
2 https://en.wikibooks.org/wiki/User:94rain
3 https://en.wikibooks.org/w/index.php%3ftitle=User:AManNamedEdwan&action=edit&redlink=1
4 https://en.wikibooks.org/w/index.php%3ftitle=User:Abie_Small&action=edit&redlink=1
5 https://en.wikibooks.org/wiki/User:Adrignola
6 https://en.wikibooks.org/w/index.php%3ftitle=User:ApathySyndicate&action=edit&redlink=1
7 https://en.wikibooks.org/wiki/User:BethNaught
8 https://en.wikibooks.org/w/index.php%3ftitle=User:Black_Walnut&action=edit&redlink=1
9 https://en.wikibooks.org/w/index.php%3ftitle=User:Bryant1410&action=edit&redlink=1
10 https://en.wikibooks.org/wiki/User:Calimo
11 https://en.wikibooks.org/w/index.php%3ftitle=User:Cedar101&action=edit&redlink=1
12 https://en.wikibooks.org/w/index.php%3ftitle=User:ChemicalyImbalanced&action=edit&redlink=1
13 https://en.wikibooks.org/w/index.php%3ftitle=User:CinoI&action=edit&redlink=1
14 https://en.wikibooks.org/wiki/User:DannyS712
15 https://en.wikibooks.org/wiki/User:DavidCary
16 https://en.wikibooks.org/w/index.php%3ftitle=User:Delapouite&action=edit&redlink=1
17 https://en.wikibooks.org/w/index.php%3ftitle=User:Djmdjm&action=edit&redlink=1
18 https://en.wikibooks.org/wiki/User:Djr13
19 https://en.wikibooks.org/w/index.php%3ftitle=User:Erikweathers&action=edit&redlink=1
20 https://en.wikibooks.org/w/index.php%3ftitle=User:Gnatmat&action=edit&redlink=1

1 Ixfd64[21]
3 J36miles[22]
38 JackPotte[23]
2 Jomegat[24]
2 KainX[25]
2 Kaliszad[26]
1 Karmaburger[27]
2 Kittycataclysm[28]
1 Koavf[29]
11 Kop[30]
1469 Larsnooden[31]
1 Laudaka[32]
1 LesmanaZimmer[33]
8 MarcGarver[34]
1 MarkAHershberger[35]
1 MathXplore[36]
2 MdsShakil[37]
1 Miimo112[38]
2 Mild Bill Hiccup[39]
3 Minorax[40]
1 Mrjulesd[41]
1 Naddy[42]
1 Neils51[43]
1 Nguyentrung86[44]
1 PAStheLoD[45]

21 https://en.wikibooks.org/wiki/User:Ixfd64
22 https://en.wikibooks.org/wiki/User:J36miles
23 https://en.wikibooks.org/wiki/User:JackPotte
24 https://en.wikibooks.org/wiki/User:Jomegat
25 https://en.wikibooks.org/w/index.php%3ftitle=User:KainX&action=edit&redlink=1
26 https://en.wikibooks.org/w/index.php%3ftitle=User:Kaliszad&action=edit&redlink=1
27 https://en.wikibooks.org/w/index.php%3ftitle=User:Karmaburger&action=edit&redlink=1
28 https://en.wikibooks.org/wiki/User:Kittycataclysm
29 https://en.wikibooks.org/wiki/User:Koavf
30 https://en.wikibooks.org/w/index.php%3ftitle=User:Kop&action=edit&redlink=1
31 https://en.wikibooks.org/wiki/User:Larsnooden
32 https://en.wikibooks.org/wiki/User:Laudaka
33 https://en.wikibooks.org/wiki/User:LesmanaZimmer
34 https://en.wikibooks.org/wiki/User:MarcGarver
35 https://en.wikibooks.org/wiki/User:MarkAHershberger
36 https://en.wikibooks.org/wiki/User:MathXplore
37 https://en.wikibooks.org/wiki/User:MdsShakil
38 https://en.wikibooks.org/w/index.php%3ftitle=User:Miimo112&action=edit&redlink=1
39 https://en.wikibooks.org/w/index.php%3ftitle=User:Mild_Bill_Hiccup&action=edit&redlink=1
40 https://en.wikibooks.org/wiki/User:Minorax
41 https://en.wikibooks.org/wiki/User:Mrjulesd
42 https://en.wikibooks.org/w/index.php%3ftitle=User:Naddy&action=edit&redlink=1
43 https://en.wikibooks.org/w/index.php%3ftitle=User:Neils51&action=edit&redlink=1
44 https://en.wikibooks.org/w/index.php%3ftitle=User:Nguyentrung86&action=edit&redlink=1
45 https://en.wikibooks.org/w/index.php%3ftitle=User:PAStheLoD&action=edit&redlink=1

1 Payerle[46]
2 Pi zero[47]
1 QdREFy6[48]
2 R. Henrik Nilsson[49]
1 Razr Nation[50]
1 Rhkramer[51]
1 Rich Farmbrough[52]
3 SHB2000[53]
11 Salva~enwikibooks[54]
1 SaschaSchwarz01[55]
1 SchreiberBike[56]
3 ShakespeareFan00[57]
1 Sietse Snel[58]
3 Splurben[59]
4 Swapnil durgade[60]
17 Tech201805[61]
1 Thenub314[62]
1 Usr-local-dick[63]
2 WereSpielChequers[64]
1 Whiteknight[65]
1 Zong~enwikibooks[66]

46 https://en.wikibooks.org/w/index.php%3ftitle=User:Payerle&action=edit&redlink=1
47 https://en.wikibooks.org/wiki/User:Pi_zero
48 https://en.wikibooks.org/w/index.php%3ftitle=User:QdREFy6&action=edit&redlink=1
49 https://en.wikibooks.org/wiki/User:R._Henrik_Nilsson
50 https://en.wikibooks.org/wiki/User:Razr_Nation
51 https://en.wikibooks.org/w/index.php%3ftitle=User:Rhkramer&action=edit&redlink=1
52 https://en.wikibooks.org/wiki/User:Rich_Farmbrough
53 https://en.wikibooks.org/wiki/User:SHB2000
54 https://en.wikibooks.org/w/index.php%3ftitle=User:Salva~enwikibooks&action=edit&redlink=1
55 https://en.wikibooks.org/w/index.php%3ftitle=User:SaschaSchwarz01&action=edit&redlink=1
56 https://en.wikibooks.org/wiki/User:SchreiberBike
57 https://en.wikibooks.org/wiki/User:ShakespeareFan00
58 https://en.wikibooks.org/w/index.php%3ftitle=User:Sietse_Snel&action=edit&redlink=1
59 https://en.wikibooks.org/w/index.php%3ftitle=User:Splurben&action=edit&redlink=1
60 https://en.wikibooks.org/wiki/User:Swapnil_durgade
61 https://en.wikibooks.org/wiki/User:Tech201805
62 https://en.wikibooks.org/wiki/User:Thenub314
63 https://en.wikibooks.org/w/index.php%3ftitle=User:Usr-local-dick&action=edit&redlink=1
64 https://en.wikibooks.org/wiki/User:WereSpielChequers
65 https://en.wikibooks.org/wiki/User:Whiteknight
66 https://en.wikibooks.org/w/index.php%3ftitle=User:Zong~enwikibooks&action=edit&redlink=1

List of Figures

- GFDL: Gnu Free Documentation License. http://www.gnu.org/licenses/fdl.html
- cc-by-sa-4.0: Creative Commons Attribution ShareAlike 4.0 License. https://creativecommons.org/licenses/by-sa/4.0/deed.en
- cc-by-sa-3.0: Creative Commons Attribution ShareAlike 3.0 License. http://creativecommons.org/licenses/by-sa/3.0/
- cc-by-sa-2.5: Creative Commons Attribution ShareAlike 2.5 License. http://creativecommons.org/licenses/by-sa/2.5/
- cc-by-sa-2.0: Creative Commons Attribution ShareAlike 2.0 License. http://creativecommons.org/licenses/by-sa/2.0/
- cc-by-sa-1.0: Creative Commons Attribution ShareAlike 1.0 License. http://creativecommons.org/licenses/by-sa/1.0/
- cc-by-1.0: Creative Commons Attribution 1.0 License. https://creativecommons.org/licenses/by/1.0/deed.en
- cc-by-2.0: Creative Commons Attribution 2.0 License. http://creativecommons.org/licenses/by/2.0/
- cc-by-2.5: Creative Commons Attribution 2.5 License. http://creativecommons.org/licenses/by/2.5/deed.en
- cc-by-3.0: Creative Commons Attribution 3.0 License. http://creativecommons.org/licenses/by/3.0/deed.en
- cc-by-4.0: Creative Commons Attribution 4.0 License. https://creativecommons.org/licenses/by/4.0/deed.de
- GPL: GNU General Public License. http://www.gnu.org/licenses/gpl-2.0.txt
- LGPL: GNU Lesser General Public License. http://www.gnu.org/licenses/lgpl.html
- PD: This image is in the public domain.
- ATTR: The copyright holder of this file allows anyone to use it for any purpose, provided that the copyright holder is properly attributed. Redistribution, derivative work, commercial use, and all other use is permitted.
- EURO: This is the common (reverse) face of a euro coin. The copyright on the design of the common face of the euro coins belongs to the European Commission. Authorised

is reproduction in a format without relief (drawings, paintings, films) provided they are not detrimental to the image of the euro.

- LFK: Lizenz Freie Kunst. `http://artlibre.org/licence/lal/de`
- CFR: Copyright free use.
- EPL: Eclipse Public License. `http://www.eclipse.org/org/documents/epl-v10.php`

Copies of the GPL, the LGPL as well as a GFDL are included in chapter Licenses[67]. Please note that images in the public domain do not require attribution. You may click on the image numbers in the following table to open the webpage of the images in your webbrower.

67 Chapter 26 on page 245

1	The people from the Tango! project[68]	PD
2	Larsnooden, GPSLeo, BotMultichillT, Jdx, YaCBot, YiFeiBot	CC-BY-4.0
3	Larsnooden, From Hill To Shore, BotMultichill, YaCBot, YiFeiBot	CC-BY-SA-4.0

68 http://tango.freedesktop.org/The_People

26 Licenses

26.1 GNU GENERAL PUBLIC LICENSE

Version 3, 29 June 2007

Copyright © 2007 Free Software Foundation, Inc. <http://fsf.org/>

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed. Preamble

The GNU General Public License is a free, copyleft license for software and other kinds of works.

The licenses for most software and other practical works are designed to take away your freedom to share and change the works. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change all versions of a program–to make sure it remains free software for all its users. We, the Free Software Foundation, use the GNU General Public License for most of our software; it applies also to any other work released this way by its authors. You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for them if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs, and that you know you can do these things.

To protect your rights, we need to prevent others from denying you these rights or asking you to surrender the rights. Therefore, you have certain responsibilities if you distribute copies of the software, or if you modify it: responsibilities to respect the freedom of others.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must pass on to the recipients the same freedoms that you received. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

Developers that use the GNU GPL protect your rights with two steps: (1) assert copyright on the software, and (2) offer you this License giving you legal permission to copy, distribute and/or modify it.

For the developers' and authors' protection, the GPL clearly explains that there is no warranty for this free software. For both users' and authors' sake, the GPL requires that modified versions be marked as changed, so that their problems will not be attributed erroneously to authors of previous versions.

Some devices are designed to deny users access to install or run modified versions of the software inside them, although the manufacturer can do so. This is fundamentally incompatible with the aim of protecting users' freedom to change the software. The systematic pattern of such abuse occurs in the area of products for individuals to use, which is precisely where it is most unacceptable. Therefore, we have designed this version of the GPL to prohibit the practice for those products. If such problems arise substantially in other domains, we stand ready to extend this provision to those domains in future versions of the GPL, as needed to protect the freedom of users.

Finally, every program is threatened constantly by software patents. States should not allow patents to restrict development and use of software on general-purpose computers, but in those that do, we wish to avoid the special danger that patents applied to a free program could make it effectively proprietary. To prevent this, the GPL assures that patents cannot be used to render the program non-free.

The precise terms and conditions for copying, distribution and modification follow. TERMS AND CONDITIONS 0. Definitions.

"This License" refers to version 3 of the GNU General Public License.

"Copyright" also means copyright-like laws that apply to other kinds of works, such as semiconductor masks.

"The Program" refers to any copyrightable work licensed under this License. Each licensee is addressed as "you". "Licensees" and "recipients" may be individuals or organizations.

To "modify" a work means to copy from or adapt all or part of the work in a fashion requiring copyright permission, other than the making of an exact copy. The resulting work is called a "modified version" of the earlier work or a work "based on" the earlier work.

A "covered work" means either the unmodified Program or a work based on the Program.

To "propagate" a work means to do anything with it that, without permission, would make you directly or secondarily liable for infringement under applicable copyright law, except executing it on a computer or modifying a private copy. Propagation includes copying, distribution (with or without modification), making available to the public, and in some countries other activities as well.

To "convey" a work means any kind of propagation that enables other parties to make or receive copies. Mere interaction with a user through a computer network, with no transfer of a copy, is not conveying.

An interactive user interface displays "Appropriate Legal Notices" to the extent that it includes a convenient and prominently visible feature that (1) displays an appropriate copyright notice, and (2) tells the user that there is no warranty for the work (except to the extent that warranties are provided), that licensees may convey the work under this License, and how to view a copy of this License. If the interface presents a list of user commands or options, such as a menu, a prominent item in the list meets this criterion. 1. Source Code.

The "source code" for a work means the preferred form of the work for making modifications to it. "Object code" means any non-source form of a work.

A "Standard Interface" means an interface that either is an official standard defined by a recognized standards body, or, in the case of interfaces specified for a particular programming language, one that is widely used among developers working in that language.

The "System Libraries" of an executable work include anything, other than the work as a whole, that (a) is included in the normal form of packaging a Major Component, but which is not part of that Major Component, and (b) serves only to enable use of the work with that Major Component, or to implement a Standard Interface for which an implementation is available to the public in source code form. A "Major Component", in this context, means a major essential component (kernel, window system, and so on) of the specific operating system (if any) on which the executable work runs, or a compiler used to produce the work, or an object code interpreter used to run it.

The "Corresponding Source" for a work in object code form means all the source code needed to generate, install, and (for an executable work) run the object code and to modify the work, including scripts to control those activities. However, it does not include the work's System Libraries, or general-purpose tools or generally available free programs which are used unmodified in performing those activities but which are not part of the work. For example, Corresponding Source includes interface definition files associated with source files for the work, and the source code for shared libraries and dynamically linked subprograms that the work is specifically designed to require, such as by intimate data communication or control flow between those subprograms and other parts of the work.

The Corresponding Source need not include anything that users can regenerate automatically from other parts of the Corresponding Source.

The Corresponding Source for a work in source code form is that same work. 2. Basic Permissions.

All rights granted under this License are granted for the term of copyright on the Program, and are irrevocable provided the stated conditions are met. This License explicitly affirms your unlimited permission to run the unmodified Program. The output from running a covered work is covered by this License only if the output, given its content, constitutes a covered work. This License acknowledges your rights of fair use or other equivalent, as provided by copyright law.

You may make, run and propagate covered works that you do not convey, without conditions so long as your license otherwise remains in force. You may convey covered works to others for the sole purpose of having them make modifications exclusively for you, or provide you with facilities for running those works, provided that you comply with the terms of this License in conveying all material for which you do not control copyright. Those thus making or running the covered works for you must do so exclusively on your behalf, under your direction and control, on terms that prohibit them from making any copies of your copyrighted material outside their relationship with you.

Conveying under any other circumstances is permitted solely under the conditions stated below. Sublicensing is not allowed; section 10 makes it unnecessary. 3. Protecting Users' Legal Rights From Anti-Circumvention Law.

No covered work shall be deemed part of an effective technological measure under any applicable law fulfilling obligations under article 11 of the WIPO copyright treaty adopted on 20 December 1996, or similar laws prohibiting or restricting circumvention of such measures.

When you convey a covered work, you waive any legal power to forbid circumvention of technological measures to the extent such circumvention is effected by exercising rights under this License with respect to the covered work, and you disclaim any intention to limit operation or modification of the work as a means of enforcing, against the work's users, your or third parties' legal rights to forbid circumvention of technological measures. 4. Conveying Verbatim Copies.

You may convey verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice; keep intact all notices stating that this License and any non-permissive terms added in accord with section 7 apply to the code; keep intact all notices of the absence of any warranty; and give all recipients a copy of this License along with the Program.

You may charge any price or no price for each copy that you convey, and you may offer support or warranty protection for a fee. 5. Conveying Modified Source Versions.

You may convey a work based on the Program, or the modifications to produce it from the Program, in the form of source code under the terms of section 4, provided that you also meet all of these conditions:

* a) The work must carry prominent notices stating that you modified it, and giving a relevant date. * b) The work must carry prominent notices stating that it is released under this License and any conditions added under section 7. This requirement modifies the requirement in section 4 to "keep intact all notices". * c) You must license the entire work, as a whole, under this License to anyone who comes into possession of a copy. This License will therefore apply, along with any applicable section 7 additional terms, to the whole of the work, and all its parts, regardless of how they are packaged. This License gives no permission to license the work in any other way, but it does not invalidate such permission if you have separately received it. * d) If the work has interactive user interfaces, each must display Appropriate Legal Notices; however, if the Program has interactive interfaces that do not display Appropriate Legal Notices, your work need not make them do so.

A compilation of a covered work with other separate and independent works, which are not by their nature extensions of the covered work, and which are not combined with it such as to form a larger program, in or on a volume of a storage or distribution medium, is called an "aggregate" if the compilation and its resulting copyright are not used to limit the access or legal rights of the compilation's users beyond what the individual works permit. Inclusion of a covered work in an aggregate does not cause this License to apply to the other parts of the aggregate. 6. Conveying Non-Source Forms.

You may convey a covered work in object code form under the terms of sections 4 and 5, provided that you also convey the machine-readable Corresponding Source under the terms of this License, in one of these ways:

* a) Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by the Corresponding Source fixed on a durable physical medium customarily used for software interchange. * b) Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by a written offer, valid for at least three years and valid for as long as you offer spare parts or customer support for that product model, to give anyone who possesses the object code either (1) a copy of the Corresponding Source for all the software in the product that is covered by this License, on a durable physical medium customarily used for software interchange, for a price no more than your reasonable cost of physically performing this conveying of source, or (2) access to copy the Corresponding Source from a network server at no charge. * c) Convey individual copies of the object code with a copy of the written offer to provide the Corresponding Source. This alternative is allowed only occasionally and noncommercially, and only if you received the object code with such an offer, in accord with subsection 6b. * d) Convey the object code by offering access from a designated place (gratis or for a charge), and offer equivalent access to the Corresponding Source in the same way through the same place at no further charge. You need not require recipients to copy the Corresponding Source along with the object code. If the place to copy the object code is a network server, the Corresponding Source may be on a different server (operated by you or a third party) that supports equivalent copying facilities, provided you maintain clear directions next to the object code saying where to find the Corresponding Source. Regardless of what server hosts the Corresponding Source, you remain obligated to ensure that it is available for as long as needed to satisfy these requirements. * e) Convey the object code using peer-to-peer transmission, provided you inform other peers where the object code and Corresponding Source of the work are being offered to the general public at no charge under subsection 6d.

A separable portion of the object code, whose source code is excluded from the Corresponding Source as a System Library, need not be included in conveying the object code work.

A "User Product" is either (1) a "consumer product", which means any tangible personal property which is normally used for personal, family, or household purposes, or (2) anything designed or sold for incorporation into a dwelling. In determining whether a product is a consumer product, doubtful cases shall be resolved in favor of coverage. For a particular product received by a particular user, "normally used" refers to a typical or common use of that class of product, regardless of the status of the particular user or of the way in which the particular user actually uses, or expects or is expected to use, the product. A product is a consumer product regardless of whether the product has substantial commercial, industrial or non-consumer uses, unless such uses represent the only significant mode of use of the product.

"Installation Information" for a User Product means any methods, procedures, authorization keys, or other information required to install and execute modified versions of a covered work in that User Product from a modified version of its Corresponding Source. The information must suffice to ensure that the continued functioning of the modified object code is in no case prevented or interfered with solely because modification has been made.

If you convey an object code work under this section in, or with, or specifically for use in, a User Product, and the conveying occurs as part of a transaction in which the right of possession and use of the User Product is transferred to the recipient in perpetuity or for a fixed term (regardless of how the transaction is characterized), the Corresponding Source conveyed under this section must be accompanied by the Installation Information. But this requirement does not apply if neither you nor any third party retains the ability to install modified object code on the User Product (for example, the work has been installed in ROM).

The requirement to provide Installation Information does not include a requirement to continue to provide support service, warranty, or updates for a work that has been modified or installed by the recipient, or for the User Product in which it has been modified or installed. Access to a network may be denied when the modification itself materially and adversely affects the operation of the network or violates the rules and protocols for communication across the network.

Corresponding Source conveyed, and Installation Information provided, in accord with this section must be in a format that is publicly documented (and with an implementation available to the public in source code form), and must require no special password or key for unpacking, reading or copying. 7. Additional Terms.

"Additional permissions" are terms that supplement the terms of this License by making exceptions from one or more of its conditions. Additional permissions that are applicable to the entire Program shall be treated as though they were included in this License, to the extent that they are valid under applicable law. If additional permissions apply only to part of the Program, that part may be used separately under those permissions, but the entire Program remains governed by this License without regard to the additional permissions.

When you convey a copy of a covered work, you may at your option remove any additional permissions from that copy, or from any part of it. (Additional permissions may be written to require their own removal in certain cases when you modify the work.) You may place additional permissions on material, added by you to a covered work, for which you have or can give appropriate copyright permission.

Notwithstanding any other provision of this License, for material you add to a covered work, you may (if authorized by the copyright holders of that material) supplement the terms of this License with terms:

* a) Disclaiming warranty or limiting liability differently from the terms of sections 15 and 16 of this License; or * b) Requiring preservation of specified reasonable legal notices or author attributions in that material or in the Appropriate Legal Notices displayed by works containing it; or * c) Prohibiting misrepresentation of the origin of that material, or requiring that modified versions of such material be marked in reasonable ways as different from the original version; or * d) Limiting the use for publicity purposes of names of licensors or authors of the material; or * e) Declining to grant rights under trademark law for use of some trade names, trademarks, or service marks; or * f) Requiring indemnification of licensors and authors of that material by anyone who conveys the material (or modified versions of it) with contractual assumptions of liability to the recipient, for any liability that these contractual assumptions directly impose on those licensors and authors.

All other non-permissive additional terms are considered "further restrictions" within the meaning of section 10. If the Program as you received it, or any part of it, contains a notice stating that it is governed by this License along with a term that is a further restriction, you may remove that term. If a license document contains a further restriction but permits relicensing or conveying under this License, you may add to a covered work material governed by the terms of that license document, provided that the further restriction does not survive such relicensing or conveying.

If you add terms to a covered work in accord with this section, you must place, in the relevant source files, a statement of the additional terms that apply to those files, or a notice indicating where to find the applicable terms.

Additional terms, permissive or non-permissive, may be stated in the form of a separately written license, or stated as exceptions; the above requirements apply either way. 8. Termination.

You may not propagate or modify a covered work except as expressly provided under this License. Any attempt otherwise to propagate or modify it is void, and will automatically terminate your rights under this License (including any patent licenses granted under the third paragraph of section 11).

However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, you do not qualify to receive new licenses for the same material under section 10. 9. Acceptance Not Required for Having Copies.

You are not required to accept this License in order to receive or run a copy of the Program. Ancillary propagation of a covered work occurring solely as a consequence of using peer-to-peer transmission to receive a copy likewise does not require acceptance. However, nothing other than this License grants you permission to propagate or modify any covered work. These actions infringe copyright if you do not accept this License. Therefore, by modifying or propagating a covered work, you indicate your acceptance of this License to do so. 10. Automatic Licensing of Downstream Recipients.

Each time you convey a covered work, the recipient automatically receives a license from the original licensors, to run, modify and propagate that work, subject to this License. You are not responsible for enforcing compliance by third parties with this License.

An "entity transaction" is a transaction transferring control of an organization, or substantially all assets of one, or subdividing an organization, or merging organizations. If propagation of a covered work results from an entity transaction, each party to that transaction who receives a copy of the work also receives whatever licenses to the work the party's predecessor in interest had or could give under the previous paragraph, plus a right to possession of the Corresponding Source of the work from the predecessor in interest, if the predecessor has it or can get it with reasonable efforts.

You may not impose any further restrictions on the exercise of the rights granted or affirmed under this License. For example, you may not impose a license fee, royalty, or other charge for exercise of rights granted under this License, and you may not initiate litigation (including a cross-claim or counterclaim in a lawsuit) alleging that any patent claim is infringed by making, using, selling, offering for sale, or importing the Program or any portion of it. 11. Patents.

A "contributor" is a copyright holder who authorizes use under this License of the Program or a work on which the Program is based. The work thus licensed is called the contributor's "contributor version".

A contributor's "essential patent claims" are all patent claims owned or controlled by the contributor, whether already acquired or hereafter acquired, that would be infringed by some manner, permitted by this License, of making, using, or selling its contributor version, but do not include claims that would be infringed only as a consequence of further modification of the contributor version. For purposes of this definition, "control" includes the right to grant patent sublicenses in a manner consistent with the requirements of this License.

Each contributor grants you a non-exclusive, worldwide, royalty-free patent license under the contributor's essential patent claims, to make, use, sell, offer for sale, import and otherwise run, modify and propagate the contents of its contributor version.

In the following three paragraphs, a "patent license" is any express agreement or commitment, however denominated, not to enforce a patent (such as an express permission to practice a patent or covenant not to sue for patent infringement). To "grant" such a patent license to a party means to make such an agreement or commitment not to enforce a patent against the party.

If you convey a covered work, knowingly relying on a patent license, and the Corresponding Source of the work is not available for anyone to copy, free of charge and under the terms of this License, through a publicly available network server or other readily accessible means, then you must either (1) cause the Corresponding Source to be so available, or (2) arrange to deprive yourself of the benefit of the patent license for this particular work, or (3) arrange, in a manner consistent with the requirements of this License, to extend the patent license to downstream recipients. "Knowingly relying" means you have actual knowledge that, but for the patent license, your conveying the covered work in a country, or your recipient's use of the covered work in a country, would infringe one or more identifiable patents in that country that you have reason to believe are valid.

If, pursuant to or in connection with a single transaction or arrangement, you convey, or propagate by procuring conveyance of, a covered work, and grant a patent license to some of the parties receiving the covered work authorizing them to use, propagate, modify or convey a specific copy of the covered work, then the patent license you grant is automatically extended to all recipients of the covered work and works based on it.

A patent license is "discriminatory" if it does not include within the scope of its coverage, prohibits the exercise of, or is conditioned on the non-exercise of one or more of the rights that are specifically granted under this License. You may not convey a covered work if you are a party to an arrangement with a third party that is in the business of distributing software, under which you make payment to the third party based on the extent of your activity of conveying the work, and under which the third party grants, to any of the parties who would receive the covered work from you, a discriminatory patent license (a) in connection with copies of the covered work conveyed by you (or copies made from those copies), or (b) primarily for and in connection with specific products or compilations that contain the covered work, unless you entered into that arrangement, or that patent license was granted, prior to 28 March 2007.

Nothing in this License shall be construed as excluding or limiting any implied license or other defenses to infringement that may otherwise be available to you under applicable patent law. 12. No Surrender of Others' Freedom.

If conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot convey a covered work so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not convey it at all. For example, if you agree to terms that obligate you to collect a royalty for further conveying from those to whom you convey the Program, the only way you could satisfy both

those terms and this License would be to refrain entirely from conveying the Program. 13. Use with the GNU Affero General Public License.

Notwithstanding any other provision of this License, you have permission to link or combine any covered work with a work licensed under version 3 of the GNU Affero General Public License into a single combined work, and to convey the resulting work. The terms of this License will continue to apply to the part which is the covered work, but the special requirements of the GNU Affero General Public License, section 13, concerning interaction through a network will apply to the combination as such. 14. Revised Versions of this License.

The Free Software Foundation may publish revised and/or new versions of the GNU General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies that a certain numbered version of the GNU General Public License "or any later version" applies to it, you have the option of following the terms and conditions either of that numbered version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of the GNU General Public License, you may choose any version ever published by the Free Software Foundation.

If the Program specifies that a proxy can decide which future versions of the GNU General Public License can be used, that proxy's public statement of acceptance of a version permanently authorizes you to choose that version for the Program.

Later license versions may give you additional or different permissions. However, no additional obligations are imposed on any author or copyright holder as a result of your choosing to follow a later version. 15. Disclaimer of Warranty.

THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION. 16. Limitation of Liability.

IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MODIFIES AND/OR CONVEYS THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. 17. Interpretation of Sections 15 and 16.

If the disclaimer of warranty and limitation of liability provided above cannot be given local legal effect according to their terms, reviewing courts shall apply local law that most closely approximates an absolute waiver of all civil liability in connection with the Program, unless a warranty or assumption of liability accompanies a copy of the Program in return for a fee.

END OF TERMS AND CONDITIONS How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively state the exclusion of warranty; and each file should have at least the "copyright" line and a pointer to where the full notice is found.

<one line to give the program's name and a brief idea of what it does.> Copyright (C) <year> <name of author>

This program is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program. If not, see <http://www.gnu.org/licenses/>.

Also add information on how to contact you by electronic and paper mail.

If the program does terminal interaction, make it output a short notice like this when it starts in an interactive mode:

<program> Copyright (C) <year> <name of author> This program comes with ABSOLUTELY NO WARRANTY; for details type 'show w'. This is free software, and you are welcome to redistribute it under certain conditions; type 'show c' for details.

The hypothetical commands 'show w' and 'show c' should show the appropriate parts of the General Public License. Of course, your program's commands might be different; for a GUI interface, you would use an "about box".

You should also get your employer (if you work as a programmer) or school, if any, to sign a "copyright disclaimer" for the program, if necessary. For more information on this, and how to apply and follow the GNU GPL, see <http://www.gnu.org/licenses/>.

The GNU General Public License does not permit incorporating your program into proprietary programs. If your program is a subroutine library, you may consider it more useful to permit linking proprietary applications with the library. If this is what you want to do, use the GNU Lesser General Public License instead of this License. But first, please read <http://www.gnu.org/philosophy/why-not-lgpl.html>.

26.2 GNU Free Documentation License

Version 1.3, 3 November 2008

Copyright © 2000, 2001, 2002, 2007, 2008 Free Software Foundation, Inc. <http://fsf.org/>

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed. 0. PREAMBLE

The purpose of this License is to make a manual, textbook, or other functional and useful document "free" in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference. 1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you". You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A "Secondary Section" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "Invariant Sections" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The "Cover Texts" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A "Transparent" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not "Transparent" is called "Opaque".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The "Title Page" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

The "publisher" means any person or entity that distributes copies of the Document to the public.

A section "Entitled XYZ" means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as "Acknowledgements", "Dedications", "Endorsements", or "History".) To "Preserve the Title" of such a section when you modify the Document means that it remains a section "Entitled XYZ" according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License. 2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies. 3. COPYING IN QUANTITY

If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document. 4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

* A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission. * B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement. * C. State on the Title page the name of the publisher of the Modified Version, as the publisher. * D. Preserve all the copyright notices of the Document. * E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices. * F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below. * G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice. * H. Include an unaltered copy of this License. * I. Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence. * J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission. * K. For any section Entitled "Acknowledgements" or "Dedications", Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein. * L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles. * M. Delete any section Entitled "Endorsements". Such a section may not be included in the Modified Version. * N. Do not retitle any existing section to be Entitled "Endorsements" or to conflict in title with any Invariant Section. * O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section Entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties—for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version. 5. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections Entitled "History" in the various original documents, forming one section Entitled "History"; likewise combine any sections Entitled "Acknowledgements", and any sections Entitled "Dedications". You must delete all sections Entitled "Endorsements". 6. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document. 7. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an "aggregate" if the copyright resulting from the compilation is not used to limit the legal rights of the compilation's users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document's Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate. 8. TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

If a section in the Document is Entitled "Acknowledgements", "Dedications", or "History", the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title. 9. TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense, or distribute it is void, and will automatically terminate your rights under this License.

However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, receipt of a copy of some or all of the same material does not give you any rights to use it. 10. FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See http://www.gnu.org/copyleft/.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation. If the Document specifies that a proxy can decide which future versions of this License can be used, that proxy's public statement of acceptance of a version permanently authorizes you to choose that version for the Document. 11. RELICENSING

"Massive Multiauthor Collaboration Site" (or "MMC Site") means any World Wide Web server that publishes copyrightable works and also provides prominent facilities for anybody to edit those works. A public wiki that anybody can edit is an example of such a server. A "Massive Multiauthor Collaboration" (or "MMC") contained in the site means any set of copyrightable works thus published on the MMC site.

"CC-BY-SA" means the Creative Commons Attribution-Share Alike 3.0 license published by Creative Commons Corporation, a not-for-profit corporation with a principal place of business in San Francisco, California, as well as future copyleft versions of that license published by that same organization.

"Incorporate" means to publish or republish a Document, in whole or in part, as part of another Document.

An MMC is "eligible for relicensing" if it is licensed under this License, and if all works that were first published under this License somewhere other than this MMC, and subsequently incorporated in whole or in part into the MMC, (1) had no cover texts or invariant sections, and (2) were thus incorporated prior to November 1, 2008.

The operator of an MMC Site may republish an MMC contained in the site under CC-BY-SA on the same site at any time before August 1, 2009, provided the MMC is eligible for relicensing. ADDENDUM: How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

Copyright (C) YEAR YOUR NAME. Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.3 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled "GNU Free Documentation License".

If you have Invariant Sections, Front-Cover Texts and Back-Cover Texts, replace the "with ... Texts." line with this:

with the Invariant Sections being LIST THEIR TITLES, with the Front-Cover Texts being LIST, and with the Back-Cover Texts being LIST.

If you have Invariant Sections without Cover Texts, or some other combination of the three, merge those two alternatives to suit the situation.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.

26.3 GNU Lesser General Public License

GNU LESSER GENERAL PUBLIC LICENSE

Version 3, 29 June 2007

Copyright © 2007 Free Software Foundation, Inc. <http://fsf.org/>

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

This version of the GNU Lesser General Public License incorporates the terms and conditions of version 3 of the GNU General Public License, supplemented by the additional permissions listed below. 0. Additional Definitions.

As used herein, "this License" refers to version 3 of the GNU Lesser General Public License, and the "GNU GPL" refers to version 3 of the GNU General Public License.

"The Library" refers to a covered work governed by this License, other than an Application or a Combined Work as defined below.

An "Application" is any work that makes use of an interface provided by the Library, but which is not otherwise based on the Library. Defining a subclass of a class defined by the Library is deemed a mode of using an interface provided by the Library.

A "Combined Work" is a work produced by combining or linking an Application with the Library. The particular version of the Library with which the Combined Work was made is also called the "Linked Version".

The "Minimal Corresponding Source" for a Combined Work means the Corresponding Source for the Combined Work, excluding any source code for portions of the Combined Work that, considered in isolation, are based on the Application, and not on the Linked Version.

The "Corresponding Application Code" for a Combined Work means the object code and/or source code for the Application, including any data and utility programs needed for reproducing the Combined Work from the Application, but excluding the System Libraries of the Combined Work. 1. Exception to Section 3 of the GNU GPL.

You may convey a covered work under sections 3 and 4 of this License without being bound by section 3 of the GNU GPL. 2. Conveying Modified Versions.

If you modify a copy of the Library, and, in your modifications, a facility refers to a function or data to be supplied by an Application that uses the facility (other than as an argument passed when the facility is invoked), then you may convey a copy of the modified version:

* a) under this License, provided that you make a good faith effort to ensure that, in the event an Application does not supply the function or data, the facility still operates, and performs whatever part of its purpose remains meaningful, or * b) under the GNU GPL, with none of the additional permissions of this License applicable to that copy.

3. Object Code Incorporating Material from Library Header Files.

The object code form of an Application may incorporate material from a header file that is part of the Library. You may convey such object code under terms of your choice, provided that, if the incorporated material is not limited to numerical parameters, data structure layouts and accessors, or small macros, inline functions and templates (ten or fewer lines in length), you do both of the following:

* a) Give prominent notice with each copy of the object code that the Library is used in it and that the Library and its use are covered by this License. * b) Accompany the object code with a copy of the GNU GPL and this license document.

4. Combined Works.

You may convey a Combined Work under terms of your choice that, taken together, effectively do not restrict modification of the portions of the Library contained in the Combined Work and reverse engineering for debugging such modifications, if you also do each of the following:

* a) Give prominent notice with each copy of the Combined Work that the Library is used in it and that the Library and its use are covered by this License. * b) Accompany the Combined Work with a copy of the GNU GPL and this license document. * c) For a Combined Work that displays copyright notices during execution, include the copyright notice for the Library among these notices, as well as a reference directing the user to the copies of the GNU GPL and this license document. * d) Do one of the following: o 0) Convey the Minimal Corresponding Source under the terms of this License, and the Corresponding Application Code in a form suitable for, and under terms that permit, the user to recombine or relink the Application with a modified version of the Linked Version to produce a modified Combined Work, in the manner specified by section 6 of the GNU GPL for conveying Corresponding Source. o 1) Use a suitable shared library mechanism for linking with the Library. A suitable mechanism is one that (a) uses at run time a copy of the Library already present on the user's computer system, and (b) will operate properly with a modified version of the Library that is interface-compatible with the Linked Version. * e) Provide Installation Information, but only if you would otherwise be required to provide such information under section 6 of the GNU GPL, and only to the extent that such information is necessary to install and execute a modified version of the Combined Work produced by recombining or relinking the Application with a modified version of the Linked Version. (If you use option 4d0, the Installation Information must accompany the Minimal Corresponding Source and Corresponding Application Code. If you use option 4d1, you must provide the Installation Information in the manner specified by section 6 of the GNU GPL for conveying Corresponding Source.)

5. Combined Libraries.

You may place library facilities that are a work based on the Library side by side in a single library together with other library facilities that are not Applications and are not covered by this License, and convey such a combined library under terms of your choice, if you do both of the following:

* a) Accompany the combined library with a copy of the same work based on the Library, uncombined with any other library facilities, conveyed under the terms of this License. * b) Give prominent notice with the combined library that part of it is a work based on the Library, and explaining where to find the accompanying uncombined form of the same work.

6. Revised Versions of the GNU Lesser General Public License.

The Free Software Foundation may publish revised and/or new versions of the GNU Lesser General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Library as you received it specifies that a certain numbered version of the GNU Lesser General Public License "or any later version" applies to it, you have the option of following the terms and conditions either of that published version or of any later version published by the Free Software Foundation. If the Library as you received it does not specify a version number of the GNU Lesser General Public License, you may choose any version of the GNU Lesser General Public License ever published by the Free Software Foundation.

If the Library as you received it specifies that a proxy can decide whether future versions of the GNU Lesser General Public License shall apply, that proxy's public statement of acceptance of any version is permanent authorization for you to choose that version for the Library.