Secure POP via SSH mini−HOWTO

Table of Contents

Secure POP via SSH mini−HOWTO

Manish Singh, <yosh@gimp.org>

v1.0, 30 September 1998

This document explains how to set up secure POP connections using ssh.

---

1. Introduction

Normal POP mail sessions, by their very nature, are insecure. The password goes across the network in cleartext for everyone to see. Now, this may be perfectly acceptable in a trusted or firewalled environment. But on a public network, such as a university or your run−of−the−mill ISP, anyone armed with a simple network sniffer can grab your password right off the wire. This is compounded by the fact that many people set their computers to check for mail at regular intervals, so the password is sent out quite frequently, which makes it easy to sniff.

With this password, an attacker can now access your email account, which may have sensitive or private information. It is also quite common that this password is the same as the user's shell account, so there is the possibility for more damage.

By doing all POP traffic using an encrypted channel, **nothing** goes in cleartext over the network. We can use ssh's diverse methods of authentication, instead of a simple plaintext password. That is the real point of using this method: not because we get encrypted content (which is futile at this point, since it's probably gone unencrypted over several networks already before reaching your mailbox; securing those communications is the job of GNU Privacy Guard or PGP, not ssh), but the secure authentication.

There are other methods of achieving secure authentication already, such as APOP, KPOP, and IMAP. However, using ssh has the advantage that it works with normal POP configurations, without requiring special client (not all mail clients support advanced protocols) or server support (except for sshd running on the server). You mail provider may be unable or unwilling to use a more secure protocol. Besides, by using ssh you can compress the traffic too, which is a nice little extra for people with slow connections.

2. The Basic Technique

This technique relies on a fundamental feature of ssh: *port forwarding*

There are many variations on this theme, which depend on your desired mail setup. They all require ssh, which is available from http://www.ssh.fi/ and mirrors. RPMs are available at ftp://ftp.replay.com/pub/crypto/ and Debian packages are available at ftp://non−us.debian.org/debian−non−US/ (and their respective mirrors).

2.1 Setting up Port Forwarding

To start port forwarding, run the following command:

```
ssh −C −f popserver −L 11110:popserver:110 sleep 5
```

Let's take a closer look at that command:

ssh

The ssh binary itself, the magic program that does it all.

−C

This enables compression of the datastream. It's optional, but usually useful, especially for dialup users.

−f

Once ssh has done authentication and established port forwarding, fork to background so other programs can be run. Since we're just using the port forwarding features of ssh, we don't need a tty attached to it.

popserver

The POP server we're connecting to.

−L 11110:popserver:110

Forward local port 11110 to port 110 on the remote server `popserver`. We use a high local port (11110) so any user can create forwardings.

sleep 5

After ssh has forked itself into the background, it runs a command. We use `sleep` so that the connection is maintained for enough time for our mail client to setup a connection to the server. 5 seconds is usually sufficient time for this to happen.

You can use most other options to ssh when appropriate. A common setting may be a username, since it might be different on the POP server.

This *requires* sshd running on the remote server `popserver`. However, you do not need to have an active shell account there. The time it takes to print a message ``You cannot telnet here'' is enough to setup a connection.

2.2 Testing it out

Once you've figured out the details command to run to establish port forwarding, you can try it. For example:

```
$ ssh −C −f msingh@popserver −L 11110:popserver:110 sleep 1000
```

`popserver` is the ol' POP server. My username on my local machine is `manish` so I need to explicitly specify the username `msingh`. (If your local and remote usernames are the same the `msingh@` part is unnecessary.

Then it prints:

```
msingh@popserver's password:
```

And I type in my POP password (you may have different shell and POP passwords though, so use your shell one). Now we're done! So we can try:

```
$ telnet localhost 11110
```

which should print something like:

```
QUALCOMM POP v3.33 ready.
```

Woohoo! It works! The data is sent out over the network encrypted, so the only cleartext is over the loopback interfaces of my local box and the POP server.

---

3. Using it With Your Mail Software

This section describes setting up your POP client software to use the ssh forwarded connection. It's primary focus is fetchmail (ESR's excellent mail−retrieval and forwarding utility), since that is the most flexible software I have found for dealing with POP. fetchmail can be found at http://www.tuxedo.org/~esr/fetchmail/. It will do you a great service to read the excellent documentation that comes with fetchmail.

3.1 Setting up fetchmail

The following is my `.fetchmailrc`

```
defaults
        user msingh is manish
        no rewrite

poll localhost with protocol pop3 and port 11110:
        preconnect "ssh -C -f msingh@popserver -L 11110:popserver:110 sleep 5"
        password foobar;
```

Pretty simple, huh? fetchmail has a wealth of commands, but the key ones are the `preconnect` line and the `poll` option.

We're not connecting directly to the POP server, but instead localhost and port 11110. The `preconnect` does the forwarding each time fetchmail is run, leaving open the connection for 5 seconds, so fetchmail can make it's own connect. The rest fetchmail does itself.

So each time you run fetchmail, you're prompted for your ssh password for authentication. If you run fetchmail in the background (like I do), it's inconvenient to have to do that. Which brings us to the next section.

3.2 Automating it all

ssh can authenticate using many methods. One of these is an RSA public/private key pair. You can generate an authentication key for your account using `ssh-keygen`. An authetication key can have a passphrase associated with it, or the passphase can be blank. Whether you want a passphrase depends on how secure you think the account you are using locally is.

If you think your machine is secure, go ahead and have a blank passpharase. Then the above `.fetchmailrc` works just by running fetchmail. You can then run fetchmail in daemon mode when you dial up and mail is fetched automatically. You're done.

However, if you think you need a passphrase, things get more complex. ssh can run under control of an **agent**, which can register keys and authenticate whatever ssh connections are made under it. So I have this script `getmail.sh`:

```
#!/bin/sh
ssh-add
while true; do fetchmail --syslog --invisible; sleep 5m; done
```

When I dialup, I run:

```
$ ssh-agent getmail.sh
```

This prompts me for my passphrase once, then checks mail every 5 minutes. When the dialup connection is closed, I terminate ssh−agent. (This is automated in my ip−up and ip−down scripts)

3.3 Not using fetchmail

What if I can't/don't want to use fetchmail? Pine, Netscape, and some other clients have their own POP mechanisms. First, consider using fetchmail! It's far more flexible, and mail clients shouldn't be doing that kind of stuff anyway. Both Pine and Netscape can be configured to use local mail systems.

But if you must, unless your client has a preconnect feature like fetchmail, you're going to have to keep the ssh port forward active for the entire time you're connected. Which means using `sleep 100000000` to keep the connection alive. This might not go over well with your network admins.

Secondly, some clients (like Netscape) have the port number hardcoded to 110. So you need to be root to do port forwarding from privledged ports. This is also annoying. But it should work.

---

4. Miscellany

4.1 Disclaimer

There is no guarantee that this document lives up to its intended purpose. This is simply provided as a free resource. As such, the author of the information provided within cannot make any guarentee that the information is even accurate. Use at your own risk.

Cryptographic software such as ssh may be subject to certain restrictions, depending on where you live. In some countries, you must have a license to use such software. If you are unsure of your local laws, please consult someone who is familiar with your situation for more information.

The use of the information provided in this document is most likely not anticipated by your mail service provider. The author does not encourage the abuse and misuse of network services, and provides this document for informational purposes only. If you are in doubt about whether the use of these techniques falls within the service agreement of your mail provider, please clear that up beforehand.

4.2 Copyright

4.3 Acknowledgements

Special thanks goes to Seth David Schoen <schoen@uclink4.berkeley.edu>, who enlightened me in the ways of ssh port forwarding.
