Firewall Piercing mini−HOWTO

Table of Contents

Firewall Piercing mini−HOWTO

François−René Rideau, fare@tunes.org

v0.7, 4 November 2000

Directions for using ppp over ssh or telnet so as to do network stuff transparently through an Internet firewall. Also applies to VPN construction.

7. Final notes

1. Stuff

1.1 DISCLAIMER

READ THIS IMPORTANT SECTION !!!

I hereby disclaim all responsibility for *your* use of this hack. If it backfires on you in any way whatsoever, that's the breaks. Not my fault. If you don't understand the risks inherent in doing this, don't do it. If you use this hack and it allows vicious vandals to break into your company's computers and costs you your job and your company millions of dollars, well that's just tough nuggies. Don't come crying to me.

1.2 Legal Blurp

1.3 Looking for a maintainer

I haven't been actively developing this mini−HOWTO recently; in particular, the french version is lagging behind. I'm looking for a maintainer to take over this document, and maybe develop software to make it easier to pierce firewalls. I also have a lot of ideas for active maintainers to expand this HOWTO, if anyone is interested.

1.4 Credits

Even though the only thing left is the disclaimers, this document owes a lot to the Term−Firewall mini−HOWTO by Barak Pearlmutter. Barak's mini−HOWTO relies on an ancient and no−more−supported program named Term (a great program in its time, and maybe still useful in some unhappy circumstances), as well as on peculiarities of a not−so−standard telnet implementation, that is, many obsolete and non−portable facts. Nevertheless, there was a necessity for a mini−HOWTO about piercing firewalls, and despite its shortcomings, his mini−HOWTO was a model and an encouragement.

I'd also like to congratulate Lars Brinkhoff and Magnus Lundström for their fine http, mail and icmp tunnels.

2. Introduction

2.1 Foreword

This is document with a moral. And the moral is: **a firewall cannot protect a network against its own internal users, and should not even try to.**

When an internal user asks you system administrator to open an outbound port to an external machine, or an inbound port to an internal machine, then you should do it for him. Of course you should help the user to make sure that his transactions are secure, and that his software is robust. But a flat out denial of service is plain incompetence. For unless he is so firewalled as be completely cut from the outside world, with no ssh, no telnet, no web browsing, no email, no ping, no phone line, no radio, no nothing, then the user can and will use firewall piercing techniques to access the machines he wants nonetheless, and the net result for security will be an unaudited connection with the outside world. So either you trust your users, after proper training and selection, or you shouldn't grant them access to the network at all. You can and you shall protect them from the outside world, but you can't protect them from themselves.

Because there exists such things as system administrators who are either unresponsive, absent, plain incompetent, or more generally managed by incompetent people, it so happens that a user may find himself behind a firewall that he may cross, but only in awkward ways. This mini−HOWTO explains a generic and portable way to pierce tunnels into firewalls, by turning any tiny small crack into a full−fledged information superhighway, so the user can seamlessly use standard tools to access computers on the other side of the firewall. The very same technique can be used by competent system administrators to build virtual private networks (VPN).

2.2 Security issues

Of course, if your sysadm has setup a firewall s/he might have a good reason, and you may have signed an agreement to not circumvent it. On the other hand, the fact that you can use telnet, the web, e−mail, or whatever other bidirectional information flux with the outside of the firewall (which is a prerequisite for the presented hacks to work) means that you are allowed to access external systems, and the fact that you can log into a particular external system somehow means you're allowed to do it, too.

So this is all a matter of *conveniently* using legal holes in a firewall, and allow generic programs to work from there with generic protocols, as opposed to requiring special or modified (and recompiled) programs going through lots of special−purpose proxies that be misconfigured by an uncaring or incompetent sysadm, or to installing lots of special−purpose converters to access each of your usual services (like e−mail) through ways supported by the firewall (like the web).

Moreover, the use of a user−level IP emulator such as SLiRP should still prevent external attackers from piercing the firewall back in the other way, unless explicitly permitted by you (or they are clever and wicked, and root or otherwise able to spy you on the remote host).

All in all, the presented hack should be *relatively* safe. However, it all depends on the particular circumstances in which you set things up, and I can give no guarantee about this hack. Lots of things are intrinsically unsafe about any Internet connection, be it with this hack or not, so don't you assume anything is safe unless you have good reasons, and/or use some kind of encryption all the way.

Let's repeat the basics of networking security: *you cannot trust anything about a connection more than you*

trust the hosts that can handle the unencrypted data, including hosts on both ends of the connection, and all hosts that can intercept the communication, unless the communication is properly encrypted with secret keys. If you misplace your trust, your passwords may be stolen and used against you, your credit card number may be stolen and used against you, and you may be fired from your work for endangering the whole company. Tough nuggies.

To sum it up, don't use this hack unless you know what you're doing. Re−read the disclaimer above.

2.3 Other requirements

It is assumed that you know what you're doing, that you know about setting up a network connection, that in case of doubt, you will have read all relevant documentation (HOWTOs, manual pages, web pages, mailing−list archives, RFCs, courses, tutorials).

It is assumed that you have shell accounts on both sides of the firewall, that you can somehow transmit packets of information both ways across the firewall (with telnet, ssh, e−mail, and the web being the ways currently known to work), and that you can let a daemon run as a background task on the remote site (or benefit from and existing daemon, sshd, telnetd, or sendmail/procmail).

It is assumed that you'll know how to configure an IP emulator (pppd, slirp) or an Internet access daemon and its associated library (SOCKS, Term) on each side, according to your needs in terms of connectivity and to your access rights, with your recompiling some software if needed.

2.4 Downloading software

Most software named in this HOWTO should be available from your standard Linux distribution, possibly among contrib's. At least, the four first below are available in as `.rpm` and `.deb` packages. In case you want to fetch the latest sources (after all, one of the ends of the connection may not be running under Linux), use the addresses below:

- `SLiRP` can be found at http://blitzen.canberra.edu.au/slirp and/or ftp://www.ibc.wustl.edu/pub/slirp_bin/.
- `zsh` can be found at http://www.zsh.org/.
- `ppp` can be found at ftp://cs.anu.edu.au/pub/software/ppp/.
- `ssh` can be found at http://www.openssh.com/.
- `fwprc` and `cotty` can be found at http://fare.tunes.org/files/fwprc/.
- `httptunnel` can be found at http://www.nocrew.org/software/httptunnel/.
- `mailtunnel` can be found at http://www.detached.net/mailtunnel/.
- `icmptunnel` can be found at http://www.detached.net/icmptunnel/.

---

3. Understanding the problem

Understanding a problem is the first half of the path to solving it.

3.1 Giving names to things

If you want this hack to work for you, you'll have to get an idea of how it works, so that in case anything breaks, you know where to look for.

The first step toward understanding the problem is to give a name to relevant concepts.

As usual, we'll herein call "local" the client machine that decides to initiate the connection, as well as programs and files on that machine; conversely, we'll call "remote" what's on the other side of the connection, where a server runs that waits for connections.

3.2 The main problem

The main problem with firewall piercing is to create a tunnel: a continuous connection from the local machine to a remote machine on the other side of the firewall, that allows for bidirectional exchange of information. Optionally, this connection should be a secure one. The secondary problem is to transform this connection into a full IP access for normal programs to use transparently.

For the main problem, we'll assume that either (1) you can establish normal TCP/IP connections from the local side of the firewall to some port on a remote machine where a sshd runs or can be set to run, or (2) you can somehow establish a telnet connection through a telnet proxy. In case you cannot, we give you pointers to other software that allows you to pierce a tunnel accross a firewall. Although we only give a secure solution in the first case, you can hack your own secure solution in the other cases, if you understand the principle (if you don't, someone, e.g. I, can do it for you in exchange for money).

3.3 The secondary problem

For the secondary problem, IP emulators (`pppd` or `SLiRP`) are run on each side of the tunnel.

On the side that wants full IP access to the other side, you'll want to run `pppd`. On the other side, you want to run `pppd` if you also want full IP access to the first side, or `SLiRP` if you want to prevent any access. Go to your usual pppd or SLiRP documentation for more information, if you have specific needs not covered by the examples given below.

Although this is conceptually trivial, it nonetheless requires a few silly tricks, so as to work, since (a) in case you're using some kind of programmed interactive shell session to start the remote IP emulator on either side, you need to correctly synchronize the start of the IP emulator on the other side, so as not to send garbage into the shell session, and (b) IP emulators are designed to be run on a "tty" interface so you have to convert your tunnel's interface into a tty one.

Issue (a) is just your usual synchronization problem, and doesn't even exist if you use `ssh`, that transparently handles remote command launching.

Issue (b) requires the use of a simple external utility. We wrote one, `cotty` just for that purpose.

<FLAME ON>

Among the silly problems caused by `pppd` maintainers' shortmindedness, you can only run it through either a device in `/dev` or the current tty. You cannot run it through a pair of pipe (which would be the obvious design). This is fine for the remote `pppd` if any, as it can use the `telnet` or `ssh` session's tty; but for the local `pppd`, this conflicts with the possible use of `telnet` as a way to establish a connection.

Indeed, `telnet`, too wants to be on a tty; it behaves *almost* correctly with a pair of pipe, except that it will still insist on doing ioctl's to the current tty, with which it will interfere; using `telnet` without a tty also causes race conditions, so that the whole connection will fail on "slow" computers (`fwprc` 0.1 worked

perfectly on a P/MMX 233, one time out of 6 on a 6x86−P200+, and never on a 486dx2/66). All in all, when using `telnet`, you need `cotty` to run as a daemon to copy output from one tty on which runs `pppd` into another tty on which runs `telnet`, and conversely.

If I find the sucker (probably a MULTICS guy, though there must have been UNIX people stupid enough to copy the idea) who invented the principle of "tty" devices by which you read and write from a "same" pseudo−file, instead of having clean pairs of pipes, I strangle him!

</FLAME>

---

4. Secure solution: piercing using ssh

4.1 Principle

Let's assume that your site administrator allows transparent TCP connections to some port on some remote machine, (be it the standard SSH port 22, or an alternate destination port, like the HTTP port 80 or whatever), or that you somehow managed to get some port in one side of the firewall to get redirected to a port on the other side (using `httptunnel`, `mailtunnel`, `icmptunnel`, some tunnel over `telnet`, or whatelse).

Then, you can run an `sshd` on the remote port, and connect to it with an `ssh` on the local port. On both sides of the `ssh` connection, you run IP emulators (`pppd`), and there you have your VPN, Virtual Public Network, that circumvents the stupid firewall limitations, with the added bonus of being encrypted for privacy (beware: the firewall administrator still knows the other end of the tunnel, and whatever authentication information you might have sent before to run `ssh`).

The exact same technology can be used to build a VPN, Virtual Private Network, whereby you securely join physical sites into a one logical network without sacrificing security with respect to the transport network between the sites.

4.2 A sample session

Below is a sample session to integrate in a shell script (it assumes sh/bash syntax; YMMV).

Be sure to edit this into a script with the right values for your needs. Use option `−p` for `ssh` to try another port than port 22 (but then, be sure to run `sshd` on same port). You can use `slirp` on the remote end, if you are not `root` there, or simply want to screen your local network from outbound connections.

Automatic reconnection is left as an exercise to the reader.

```
REMOTE_ACCOUNT=root@remote.fqdn.tld
REMOTE_PPPD="pppd ipcp−accept−local ipcp−accept−remote"
LOCAL_PPPD="pppd silent 192.168.0.1:192.168.0.2"
cotty −d −− $LOCAL_PPPD −− ssh −t $REMOTE_ACCOUNT $REMOTE_PPPD
```

(Note: this command requires `cotty` 0.4 or later.)

---

5. Unsecure solution: piercing using telnet

5.1 Principle

If all you can do is telnet (because of a telnet proxy), then this solution might be fit for you.

The firewall−piercing program, `fwprc`, will use a "tty proxy", `cotty`, that opens two pseudo−tty devices, launches some command on each of those devices' slaves, and stubbornly copies every character that one outputs to the tty that serves as input of the other command. One command will be telnet connection to remote site, and the other will be the local `pppd`. `pppd` can then open and control the telnet session with a chat script as usual.

Actually, if your telnet proxy allows connection to an arbitrary port, and if you can reliably run a daemon on the remote host (with a cron job to relaunch it in case of breakage), then you'd better write some program that will just connect a local port to the remote one through the proxy, so you can use the above secure solution, possibly using some variant of `ssh −t −o "ProxyCommand ..."` (if you submit it to me, I'll gladly integrate such a solution to the `fwprc` distribution).

Note: if you must use the unsecure telnet−based solution, be sure that nothing lies in your target account that you want to keep secret or untampered, since the password will be sent in clear text accross the Internet.

5.2 fwprc

I wrote a very well self−documented script to pierce firewalls, `fwprc`, available from my site, together with `cotty` (which is required by `fwprc` 0.2 and later). At the time of my writing these lines, latest versions are `fwprc` 0.3e and `cotty` 0.4.

The name "fwprc" is voluntarily made unreadable and unpronounceable, so that it will confuse the incompetent paranoid sysadm who might be the cause of the firewall that annoys you (of course, there can be legitimate firewalls, too, and even indispensable ones; security is all a matter of *correct* configuration). If you must read it aloud, choose the worst way you can imagine.

CONTEST! CONTEST! Send me a `.au` audio file with a digital audio recording of how you pronounce "fwprc". The worst entry will win a free upgrade and his name on the `fwprc` 1.0 page!

I tested the program in several settings, by configuring it through resource files. But of course, by Murphy's law, it will break for you. Feel free to contribute enhancements that will make life easier to other people who'll configure it after you.

5.3 .fwprcrc

`fwprc` can be customized through a file `.fwprcrc` meant to be the same on both sides of the firewall. Having several alternate configurations to choose from is sure possible (for instance, *I* do it), and is left as an exercise to the reader.

To begin with, copy the appropriate section of `fwprc` (the previous to last) into a file named `.fwprcrc` in your home directory. Then replace variable values with stuff that fits your configuration. Finally, copy to the other host, and test.

Default behavior is to use `pppd` locally, and `slirp` remotely. To modify that, you can redefine the appropriate function in your `.fwprcrc` with such a line as:

```
remote_IP_emu () { remote_pppd }
```

Note that `SLiRP` is safer than `pppd`, and easier to have access to, since it does not require being root on the remote machine, and needn't additional firewall configuration to prevent connections from the outside world into the firewalled network. The basic functionality in `SLiRP` works quite well, but I haven't managed to get some advertised pluses to work (like run−time controllability). Of course, since it is free software, feel free to hack the source so as to actually implement or fix whichever feature you need.

---

6. Reverse piercing

6.1 Rationale

Sometimes, only one side of the firewall can launch telnet sessions into the other side; however, some means of communication is possible (typically, through e−mail). Piercing the firewall is still possible, by triggering with whatever messaging capability is available a telnet connection from the ``right'' side of the firewall to the other.

`fwprc` includes code to trigger such connections from a PGP−authentified e−mail message; all you need is add `fwprc` as a `procmail` filter to messages using the protocol, (instructions included in `fwprc` itself). Note however, that if you are to launch `pppd` with appropriate privileges, you might need create your own suid wrapper to become root. Instructions enclosed in `fwprc`.

Also, authentified trigger does not remotely mean secure connection. You should really use `ssh` (perhaps over telnet) for secure connections. And then, beware of what happens between the triggering of a telnet connection, and `ssh` taking over that connection. Contribution in that direction welcome.

6.2 Getting the triggering mail

If you are firewalled, your mail may as well be in a central server that doesn't do procmail filtering or allow telnet sessions. No problem! You can use `fetchmail` to run in daemon mode to poll and get mail to your client linux system, and/or add a cron job to automatically poll for mail every 1−5 minutes. `fetchmail` will forward mail to a local address through `sendmail`, which itself will have been configured to use `procmail` for delivery. Note that if you run `fetchmail` as a background daemon, it will lock away any other fetchmail that you'd like to run only at other times, like when you open a `fwprc`; of course, if you can also run a fetchmail daemon as a fake user. Too frequent a poll won't be nice to either the server or your host. Too infrequent a poll means you'll have to wait before the message gets read and the reverse connection gets established. I use two−minute poll frequency.

---

7. Final notes

7.1 Other settings

There are other kinds of firewalls than those that allow for direct ssh or telnet connections. As long as a continuous flow of packets may transmit information through a firewall in both directions, it is possible to pierce it; only the price of writing the piercer may be higher or lower.

In a very easy case, we saw that you can just launch `ssh` over a pty master and do some `pppd` in the slave tty. You may even want to do it without an adverse firewall, just so as to build a secure ``VPN'' (Virtual Private Network). The VPN mini−HOWTO gives all the details you need about this. We invite you, as an exercise, to modify `fwprc` so as to use this technique, or perhaps even so as to use it inside a previous non−secure `fwprc` session.

Now, if the only way through the firewall is a WWW proxy (usually, a minimum for an Internet−connected network), you might want to use Lars Brinkoff's httptunnel, a http daemon and client combination that achieves a TCP/IP tunnel connection through the proxy−friendly HTTP protocol. You should then be able to run `fwprc` (preferably over `ssh`) over that connection, although I haven't tried it yet. Could anyone test and report? Note that `httptunnel` is still under development, so you may help implement the features it currently lacks, like, having multiple connections, and/or serving fake pages so as to mislead suspicious adverse firewall administrators.

Whatever goes through your firewall, be it telnet, HTTP or other TCP/IP connections, or something real weird like DNS queries, ICMP packets, e−mail (see mailtunnel, icmptunnel), or whatelse, you can always write a tunnel client/daemon combination, and run a `ssh` and/or PPP connection through it. The performance mightn't be high, depending on the effective information communication rate after paying the overhead for coding around filters and proxies; but such a tunnel is still interesting as long as it's good enough to use `fetchmail`, `suck`, and other non−interactive programs.

If you need cross a 7−bit line, you'll want to use SLIP instead of PPP. I never tried, because lines are more or less 8−bit clean these days, but it shouldn't be difficult. If necessary, fall back to using the Term−Firewall mini−HOWTO.

If you have an 8−bit clean connection and you're root on linux both sides of the firewall, you might want to use ethertap for better performance, encapsulating raw ethernet communications on top of your connection. David Madore has written ethertap−over−TCP and ethertap−over−UDP tunneling ftp://quatramaran.ens.fr/pub/madore/misc/. There remains to write some ethertap−over−tty to combine with fwprc−like tools.

If you really need more performance than you can get while paying for a user−space sequential communication tunnel through which to run PPP, then you're in the very hard case where you might have to re−hack a weird IP stack, using (for instance) the Fox project's packet−protocol functors. You'll then achieve some direct IP−over−HTTP, IP−over−DNS, IP−over−ICMP, or such, which requires not only an elaborate protocol, but also an interface to an OS kernel, both of which are costly to implement.

7.2 HOWTO maintenance

I felt it was necessary to write it, but I don't have that much time for that, so this mini−HOWTO is very rough. Thus will it stay, until I get enough feedback so as to know what sections to enhance, or better, until someone comes and takes over maintenance for the mini−HOWTO. Feedback welcome. Help welcome. mini−HOWTO maintenance take−over welcome.

In any case, the above sections have shown many problems whose solution is just a matter of someone (you?) spending some time (or money, by hiring someone else) to sit down and write it: nothing conceptually complicated, though the details might be burdensome or tricky.

Do not hesitate to contribute more problems, and hopefully more solutions, to this mini−HOWTO.

For instance, there is some need for a section on setting up routes correctly with `fwprc`, including example use of `getroute.pl` from `/etc/ppp/ip−up`.

7.3 Related Documents

The LDP publishes many documents related to this mini−HOWTO, most notably the Linux Security Knowledge Base, the VPN HOWTO, the VPN mini−HOWTO.

Then again, when facing a problem with some program, one reflex for any Linux user should be to RTFM: Read The Fscking Manual pages for the considered programs.

7.4 Extra copy of IMPORTANT DISCLAIMER −−− BELIEVE IT!!!

> **I hereby disclaim all responsibility for *your* use of this hack. If it backfires on you in any way whatsoever, that's the breaks. Not my fault. If you don't understand the risks inherent in doing this, don't do it. If you use this hack and it allows vicious vandals to break into your company's computers and costs you your job and your company millions of dollars, well that's just tough nuggies. Don't come crying to me.**
