Firewall Piercing mini−HOWTO

Table of Contents

Firewall Piercing mini−HOWTO

François−René Rideau, `fare@tunes.org`

v0.3b, 27 November 1998

Directions for using ppp over telnet to do network stuff transparently through an Internet firewall.

5.Reverse piercing

6.Final notes

1.Stuff

1.1 DISCLAIMER

READ THIS IMPORTANT SECTION !!!

I hereby disclaim all responsibility for 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 Credits

Even though I rewrote most everything but the disclaimers, I'm indebted to Barak Pearlmutter mailto:bap@cs.unm.edu for his Term−Firewall mini−HOWTO: I think there was a necessity for a mini−HOWTO about piercing firewalls, and despite its shortcomings, his mini−HOWTO was a model and an encouragement.

2.Introduction

2.1 Foreword

Because system administrators and users have different constraints and proficiencies, 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 use standard internet tools seamlessly across such firewalls, by the use of an IP emulator over a telnet session.

It is freely inspired by the Term−Firewall mini−HOWTO by Barak Pearlmutter mailto:bap@cs.unm.edu, which relies on an ancient and no−more−supported program named Term (yet a great program at its time), as well as on peculiarities of a not−so−standard telnet implementation, that is, many obsolete and non−portable facts.

2.2 Security problems

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 telnet outside (which is a requisite 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.

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 you have shell accounts on both sides of the firewall; that you can somehow telnet (or ssh, or equivalent) from one account to the other; that you can run an IP emulator on both shell accounts; that you have programs able to use the IP connection emulated on their side. Note that any program can use the connection, in case the local emulator is pppd talking to the Linux kernel; other emulators, like Term, need recompilation and linking to a special library.

Talking about IP emulators, pppd can be found in any good Linux distribution or ftp site; so can SLiRP. If your remote shell account is user−level only, you can use SLiRP to connect.

2.4 Downloading software

Most described software should be available from your standard distribution, possibly among contrib's; at least all but the two small last ones are available in as rpm packages. In case you want to fetch the latest sources or binaries (after all, one of the ends of the connection may not be running 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.peak.org/zsh/.
- `ppp` can be found at ftp://cs.anu.edu.au/pub/software/ppp/.
- `fwprc` and `cotty` can be found at http://www.tunes.org/~fare/files/fwprc/.

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.

So we'll herein call "local" the machine that initiates 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.

3.2 The problem

The goal is to connect the input and output of a local IP emulator to the output and input respectively of a remote IP emulator.

Only the communication channels with which IP emulators interact are either direct devices (in the usual case of pppd), or the "current tty". The previous case obviously does not happen with telnet sessions. The latter is tricky, because when you launch the local emulator from the command line, the "current tty" is linked to the command−line user, not to a remote session; also, should we open a new session (local or remote) on a new terminal, we must synchronize the launching and connection of IP emulators on both sides, least one session's garbage output is going to be executed as commands on the other session, which would recursively produce more garbage.

3.3 Additional difficulty

To get the best ease of use, the local IP emulator has to provide IP to kernel networking, hence be pppd. However, pppd is dumb enough to only accept having data through /dev or thru the current tty; it must be a tty, not 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 session's tty; but for the local pppd, it sucks, as it can't launch the telnet session to connect to; hence, there must some kind of wrapper around it.

Telnet 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).

[Note: 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!]

4.The solution

4.1 Principle

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.

4.2 fwprc

I wrote a very well self−documented script to pierce firewalls, `fwprc`, available from my site http://www.tunes.org/~fare/files/fwprc/, together with `cotty` (which is required by `fwprc` 0.2 and later). At the time of my writing these lines, latest versions are `fwprc` 0.3a and `cotty` 0.3a.

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 indispensible 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.

4.3 .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. Anoter safe feature is that it will drop packets not directly coming from the connected machine (which feature becomes a misfeature if you attempt to route a subnetwork onto it with masquerading). The basic functionality in SLiRP works quite well, but I've found advertised pluses (like run−time controllability) to be deficient; of course, since it is free software, feel free to hack the source so as to actually implement whichever feature you need.

5.Reverse piercing

5.1 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(1)` filter to messages using the protocol, (instructions included in `fwprc` itself). Note however, that if you are to launch pppd with appropriate priviledges, 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.

5.2 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(1)` 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(8)`, which itself will have been configured to use `procmail(1)` for delivery. Note that if you run `fetchmail(1)` 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 unfrequent 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.

6.Final notes

6.1 Other settings

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

In a very easy case, you can just launch `ssh` over a pty, and do some `pppd` in the slave tty. `cotty` 0.3a should be able to do it, but nobody's modified `fwprc` to take it into account yet. May be tonight's exercise for you. You may even want to do it without an adverse firewall, just so as to build a secure ``VPN'' (Virtual Private Network). See the VPN mini−HOWTO about this.

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.

Now, if the only way through the firewall is a WWW proxy (usually, a minimum for an internet−connected network), you might want to write a daemon that buffers data in and out, and sends it during in HTTP connections, achieving some telnet−over−HTTP over which to run fwprc. It might be slow and not very responsive, but still good enough to use `fetchmail(1)`, `suck(1)`, and other non−interactive programs.

If you want more performance, or if the only thing that goes through unfiltered is some wierder thing even (DNS queries, ICMP packets, whatever), then you're in the very hard case where you'll have to re−hack a wierd 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 a complex protocol, but also an interface to an OS kernel, both of which are costly to implement.

By the way, if you use some Firewall−piercing HTTP daemon, don't forget to have it serve fake pages, so as to mislead suspicious adverse firewall administrators.

6.2 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. So will it stay, until I get enough feedback so as to know what sections to enhance. 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.

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

> **I hereby disclaim all responsibility for 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.**