Using Term to Pierce an Internet Firewall mini–HOWTO

Barak Pearlmutter

bap@cs.unm.edu

David C. Merrill

david@lupercalia.net

Revision History

Revision 1.1	2001–07–14	Revised by: dcm
Cleaned up a bit, reorganized a bit, converted to DocBook SGML and relicensed under GFDL.		
Revision 1.0	1996–07–15	Revised by: pb
Initial Release.		

This document explains how to use the **term** program to pierce a firewall from the inside, even without root privileges.

Term is an old program that almost no one uses anymore, because the 7–bit serial lines it is meant to cross are nowhere to be found anymore, and full IP ppp access is dirt cheap.

Table of Contents

1. Preface...1
1.1. Disclaimer...1
1.2. License...1

2. Introduction...2

3. The Basic Procedure...3

4. Detailed Directions...4

5. Multiple Term Sockets...5

6. The <Ü/.term/termrc.telnet Init File...6

7. Direction...7

8. Security...8

9. Telnet Mode...9

10. Bugs and Term Wish List...10

11. Tricks That Do Not Seem to Work...11

12. Related Resources...12

13. Acknowledgments...13

1. Preface

1.1. Disclaimer

While every precaution has been taken in the preparation of this document, the Linux Documentation Project and the author(s) assume no responsibility for errors or omissions, or for damages resulting from the use of the information contained herein.

1.2. License

2. Introduction

The **term** program is usually used to provide host−to−host services over a modem or serial line. However, sometimes it is useful to establish a term connection between two machines that communicate via telnet. The most interesting example is connecting two hosts which are separated by ethernet firewalls or SOCKS servers. Such firewalls provide facilities for establishing a telnet connection through the firewall, typically by using the SOCKS protocol, to allow inside machines to get connections out, and requiring outside users to telnet first to a gateway machine which requires a one−time password. These firewalls make it impossible to, for instance, have X clients on an inside machine communicate with an X server on an outside machine. But, by setting up a term connection, these restrictions can all be bypassed quite conveniently, at the user level.

3. The Basic Procedure

Setting up a term connection over a telnet substrate is a two−phase process. First your usual telnet client is used to set up a telnet connection and log in. Next, the telnet client is paused and control of the established telnet connection is given to term.

4. Detailed Directions

First, from a machine inside the firewall, telnet to a target machine outside the firewall and log in.

Unless you are under linux and will be using the proc filesystem (see below) make sure your shell is an sh style shell. Ie if your default shell is a csh variant, invoke telnet by:

```
setenv SHELL /bin/sh; telnet machine.outside
```

After logging in, on the remote (outside) machine invoke the command:

```
term -r -n off telnet
```

Now break back to the telnet prompt on the local (inside) machine, using `^]` or whatever, and use the telnet shell escape command `!` to invoke term:

```
telnet> ! term -n on telnet >&3 <&3
```

That's it!

If you have a variant telnet, you might have to use some other file descriptor than 3; easy to check using strace. But three seems to work on all bsd descendent telnet clients I've tried, under both SunOS 4.x and the usual linux distributions.

Some telnet clients do not have the ! shell escape command. Eg the telnet client distributed with Slackware 3.0 is one such client. The sources that the Slackware telnet client is supposedly built from

ftp://ftp.cdrom.com:/pub/linux/slackware−3.0/source/n/tcpip/NetKit−B−0.05.tar.gz

A simple solution is therefore to obtain these sources and recompile them. This unfortunately is a task I have had no luck with. Plus, if you are running from inside a SOCKS firewall, you will need a SOCKSified telnet client anyway. To that end, I was able to compile a SOCKSified telnet client from:

ftp://ftp.nec.com/pub/security/socks.cstc/socks.cstc.4.2.tar.gz

or, if you're outside the USA,

ftp://ftp.nec.com/pub/security/socks.cstc/export.socks.cstc.4.2.tar.gz

Alternatively, under linux kernels up to 1.2.13, you can pause the telnet with `^]^z`, figure out its pid, and invoke:

```
term -n on -v /proc/&,t;telnetpid>/fd/3 telnet
```

This doesn't work with kernels after 1.3.x, which closed some mysterious security hole by preventing access to these fd's by processes other than the owner process and its children.

5. Multiple Term Sockets

It is a good idea to give the term socket an explicit name. This is the `telnet`; argument in the invocations of term above. Unless you have the TERMSERVER environment variable set to telnet as appropriate, you invoke term clients with the –t switch, e.g., `trsh -t telnet`.

6. The <Ü/.term/termrc.telnet Init File

I have checked line clarity using linecheck over this medium. I expected it to be completely transparent, but it is not. However, the only bad character seems to be 255. The `Ü/.term/termrc.telnet` I use (the `.telnet` is the name of the term connection, see above) contains:

```
baudrate off
escape 255
ignore 255
timeout 600
```

Perhaps it could be improved by diddling, I am getting a throughput of only about 30k cps over a long−haul connection through a slow firewall. Ftp can move about 100k cps over the same route. A realistic baudrate might avoid some timeouts.

7. Direction

Obviously, if you are starting from outside the firewall and zitching in using a SecureID card or something, you will want to reverse the roles of the remote vs local servers given above. (If you don't understand what this means, perhaps you are not familiar enough with term to use the trick described in this file responsibly.)

8. Security

This is not much more of a vulnerability than the current possibility of having a telnet connection hijacked on an unsecured outside machine. The primary additional risk comes from people being able to use the term socket you set up without you even being aware of it. So be careful out there. (Personally, I do this with an outside machine I know to be pretty secure, namely a linux laptop I maintain myself that does not accept any incoming connections.)

Another possibility is to add

```
socket off
```

to the remote `Ü/.term/termrc.telnet` file, or

```
add "-u off"
```

to the invocation of term. This prevents the socket from being hijacked from the remote end, with only a minor loss of functionality.

---

9. Telnet Mode

Be sure the remote telnetd is not in some nasty seven−bit mode. Or if it is, you have to tell term about it when you invoke term, by adding the `−a` switch at both ends. (I sometimes use `>^] telnet> set outbin` or `set bin`, or invoke telnet with a `−8` switch to put the connection into eight−bit mode.)

10. Bugs and Term Wish List

The **linecheck** program has some problems checking telnet connections sometimes. This is sometimes because it doesn't check the return code of the `read()` call it makes. For network connections, this call to `read()` can return `-1` with an `EINTR` (interrupted) or `EAGAIN` (try again) error code. Obviously this should be checked for.

There are a number of features that could ease the use of term over telnet. These primarily relate to an assumption that influenced the design of term, namely that the connection is low bandwidth, low latency, and somewhat noisy.

A telnet connection is in general high bandwidth, high latency, and error free. This means that the connection could be better utilized if (a) the maximum window size was raised, well above the limit imposed by term's `N_PACKETS/2=16`, (b) there was an option to turn off sending and checking packet checksums, and (c) larger packets were permitted when appropriate.

Also, to enhance security, it would be nice to have a term option to log all connections through the socket it monitors to a log file, or to `stderr`, or both. This would allow one to see if one's term connection is being subverted by nasty hackers on the outside insecure machine.

---

11. Tricks That Do Not Seem to Work

Some telnet clients and servers agree to encrypt their communications, to prevent eavesdropping on the connection. Unfortunately, the hack used above (using the network connection that the telnet client has set up while the telnet client is idle) won't work in that case. Instead, one really must go through the telnet client itself, so it can do its encryption. It seems like that requires a simple hack to the telnet client itself, to add a command that runs a process with its `stdin` and `stdout` are connected to the live telnet connection. This would also be useful for various bots, so perhaps someone has already hacked it up.

12. Related Resources

A vaguely related trick is to SOCKSify one's Term library. Details, including patches to SOCKS, are available from Steven Danz *danz@wv.mentorg.com*.

13. Acknowledgments

Thanks for valuable suggestions from:

- Gary Flake *flake@scr.siemens.com*
- Bill Riemers *bcr@physics.purdue.edu*
- Greg Louis *glouis@dynamicro.on.ca*