

Slow Down Internet Worms With Tarpits

Tony Bautts 2003-08-21

Worms, worms are everywhere! The recent and prolific spread of Internet worms has yet again demonstrated the vulnerability of network hosts, and it's clear that new approaches to worm containment need to be investigated. In this article, we'll discuss a new twist on an under-utilized technology: the tarpit.

The Worms

In a nutshell, worm technology works by infecting a host and then using it to scan for more victims. The damage caused by the recent worm outbreaks isn't so much to the victim computer as it is to the networks in which they operate. The side-effect of propagation is that massive amounts of bandwidth are consumed as the infected hosts perform their scanning. The speed at which they are able to compromise new hosts grows exponentially, eventually causing a network [meltdown](#).

In the future, worms could carry more damaging payloads, doing things like deleting files, installing network sniffers, or stealing confidential files. However, there is a fine balance between being overly destructive and fast to propagate, because just like in nature, a worm or virus that kills its host too quickly cannot effectively spread.

Solutions?

Preventative measures provide the most effective protection -- in this case, patching the vulnerable systems before the worm is released. In the case of the Blaster worm, a patch was available long before the worm happened, and long enough for security experts to place informal "bets" as to when the worm would actually appear. However, in large companies and organizations with traveling users, applying and supporting a patch to systems can become somewhat of a tactical nightmare. So, given that preventative maintenance obviously isn't working, it may be necessary to begin to examine some of the other possibilities for slowing the spread of worms, once outbreaks occur.

One such solution, and the focus of this article, is the TARPIT -- which is available as a relatively new patch to the Netfilter (IPTables) firewall for Linux, in addition to being available for Windows platforms, Solaris, and OpenBSD thanks to the [LaBrea tarpit project](#) from

Hackbusters (but now hosted on Sorceforge). For simplicity, this article will focus on just the IPtables version. What the tarpit project means to IPtables users is that now, instead of simply logging and dropping packets, they can now be sent to a TARPIT.

The concept behind a tarpit is fairly simple. The connections come in, but they don't get back out. IPtables handles this by allowing a tarpitted port to accept any incoming TCP connection. When data transfer begins to occur, the TCP window size is set to zero, so no data can be transferred within the session. The connection is then held open, and any requests by the remote side to close the session are ignored. This means that the attacker must wait for the connection to timeout in order to disconnect. This kind of behavior is bad news for automated scanning tools (like worms) because they rely on a quick turnaround from their potential victims.

Some caveats before we begin

Before dropping tarpits all throughout a production environment, there are a couple of points to consider. First, the concept of a TCP session not "letting go" could be really distressing to some TCP stacks and operating systems -it's even possible that they may crash. Another potential caution is that the tarpit within IPtables is fairly new code; therefore it isn't mature enough to be considered completely stable. While tarpits can offer great functionality in slowing and tracking worms, the technology should be thoroughly evaluated before deploying it into a production environment.

Putting it to work

In order to use the IPtables tarpit, you will need the [latest patch-o-matic](#) from the Netfilter web site. Some Linux distributions, such as Gentoo, offer this patch in their standard kernel source distribution. If you do need to patch, great documentation of how to use the patch-o-matic patches for IPtables exist on their site.

Once you've patched your kernel source, you will need to enable the TARPIT option and rebuild the kernel. The new options will be found under "Networking Options -> IP: Netfilter Configuration". You will need to enable the "Packet Filtering" option to see the TARPIT option. Once you've successfully enabled the TARPIT module, rebuild the kernel and modules, and reboot if necessary.

Using a tarpit is simple. It works just like any other IPtables rule. To tarpit TCP Port 135, you

would need this rule:

```
iptables -A INPUT -p tcp -m tcp --dport 135 -j TARPIT
```

So, with this run in effect, any attempt to connect to port 135 would be welcomed, and would not let go. A tcpdump of this session looks like:

```
14:22:12.039166 10.0.0.1.4439 > 10.0.0.2.135: S 333636191:333636191(0) win 16384
<mss 1460,nop,nop,sackOK> (DF)
14:22:12.039253 10.0.0.2.135 > 10.0.0.1.4439: S 1551562378:1551562378(0) ack
333636192 win 5 (DF)
14:22:12.041942 10.0.0.1.4439 > 10.10.0.2.135: . ack 1 win 16616 (DF)
14:22:12.042022 10.10.0.2.135 > 10.10.0.1.4439: . ack 1 win 0 (DF)
```

The session is established as normal, but we can already see that the TCP window size has been set to zero. When we try sending data to the tarpitted port, nothing changes:

```
14:27:25.584968 10.0.0.1.4439 > 10.0.0.2.135: P 1:2(1) ack 1 win 16616 (DF)
14:27:25.585050 10.0.0.2.135 > 10.0.0.1.4439: . ack 1 win 0 (DF)
```

Even when we give up, and try to force a connection close (by killing the application which initiated the connection), the session will remain open until it finally times out. Any type of automated scanning tool will be forced to wait for a long time before being able to proceed.

You can use TARPIT strategically too. Having your Linux machine answer on ports which are not normally Linux ports can confuse potential attackers, and possibly cause them to waste time while enumerating your system. For example, if you want your Linux machine to look like a Windows machine, you can use these rules:

```
iptables -A INPUT -p tcp -m tcp --dport 135 -j TARPIT
iptables -A INPUT -p tcp -m tcp --dport 139 -j TARPIT
iptables -A INPUT -p tcp -m tcp --dport 1025 -j TARPIT
```

This way, now, when your host is scanned, it looks like this:

```
host# scanrand 10.10.0.156
```

```

UP:      10.10.0.156:1025  [01]   3.022s
UP:      10.10.0.156:135   [01]   3.024s
UP:      10.10.0.156:139   [01]   8.980s

```

Of course, while simply spoofing the port configuration of a Windows server won't necessarily fool a determined attacker, it can be enough to slow them down.

Another crafty and unusual way to slow down would-be attackers is to tarpit all of the ports on your machine. The easiest way to do this is to add a TARPIT rule just before the final DROP rule in your current IPTables firewall. For example, most systems would have this kind of a rule set:

```

iptables -A INPUT -p tcp -m tcp --dport 22 -j ACCEPT
iptables -A INPUT -p tcp -m tcp --dport 80 -j ACCEPT
.
.
iptables -A INPUT -p tcp -j DROP

```

If you want to have a tarpit on all unused ports, the following rule should be added before your final DROP rule:

```
iptables -A INPUT -p tcp -j TARPIT
```

This kind of a configuration will result in the following Nmap scan result:

```
Starting nmap 3.30 ( http://www.insecure.org/nmap/ ) at 2003-08-16 16:21 Local
time zone must be set--see zic manual page
```

```
Interesting ports on adsl-64-168-69-66.dsl.snfc21.pacbell.net (64.168.69.66):
```

Port	State	Service
1/tcp	open	tcpmux
2/tcp	open	compressnet
3/tcp	open	compressnet
4/tcp	open	unknown
5/tcp	open	rje
6/tcp	open	unknown
7/tcp	open	echo
8/tcp	open	unknown

```
9/tcp      open      discard
10/tcp     open      unknown
11/tcp     open      systat
12/tcp     open      unknown
13/tcp     open      daytime
14/tcp     open      unknown
15/tcp     open      netstat
16/tcp     open      unknown
17/tcp     open      qotd
18/tcp     open      msp
.
.
32787/tcp  open      sometimes-rpc27
43188/tcp  open      reachout
44334/tcp  open      tinyfw
44442/tcp  open      coldfusion-auth
44443/tcp  open      coldfusion-auth
47557/tcp  open      dbbrowse
49400/tcp  open      compaqdiag
54320/tcp  open      bo2k
61439/tcp  open      netprowler-manager
61440/tcp  open      netprowler-manager2
61441/tcp  open      netprowler-sensor
65301/tcp  open      pcanynwhere
```

No OS matches for host (test conditions non-ideal).

TCP/IP fingerprint:

(None)

Nmap run completed -- 1 IP address (1 host up) scanned in 2.883 seconds

Essentially all 65,000+ ports are displayed as open. An additional benefit is that Nmap's OS detection fails because of the unexpected amount of replies. This makes host enumeration much more difficult for potential attackers, and certainly makes finding valid services much like searching for a needle in the haystack.

Just remember, though, that like all other security systems, nothing is attacker-proof. A determined mind can still find services within a tarpit.

Having set up the tarpit, it can be usefully combined with another handy IPtables feature, string matching. This feature was designed to allow users to block packets based on strings within their payload. The idea of simply blocking or tarpiting traffic on 'wormed' ports, such as http isn't practical. Therefore, being able to differentiate worm traffic from valid http requests is critical. With string matching, signature data can be used to separate worm traffic from valid traffic and send the worms to the tarpit. Consider the following rules:

```
iptables -I INPUT -j TARPIT -p tcp -s 0.0.0.0/0 --dport 80 -m string --string "default.ida"
iptables -I INPUT -j TARPIT -p tcp -s 0.0.0.0/0 --dport 80 -m string --string "cmd.exe"
```

They will take any traffic containing "cmd.exe" or "default.ida" on port 80 and send them to the tarpit. So, any CodeRed or Nimda traffic received on this host, would get stuck in the tarpit.

What works well for one host should work well for many, right? It is also possible to have a Linux machine answer requests for all ports and IPs not currently in use. In doing this, we can effectively slow a worm's progress through an entire subnet, giving extra time to respond to the incident. For example, let's say that ZZZ Inc wasn't using the 10.10.0.0/24 internal network block. They could turn this unused space into a tarpit network by directing traffic destined to this network to a Linux machine running a tarpit. Then, using two IPtables rules:

```
iptables -A FORWARD -p tcp -j TARPIT
iptables -A FORWARD -j DROP
```

Any traffic bound for that subnet would be significantly impeded. Using the TARPIT target in this way is very similar to the way the LaBrea project worked. Unfortunately, LaBrea's future is unclear, due to pending Illinois legislation which could make such work illegal. So, for the time being, if you would like have LaBrea-like functionality, IPTables tarpit will do nicely.

Conclusion

The spread of worms on the Internet continues to be a major issue, affecting hundreds of thousands or even millions of network hosts around the world. While patches are often available

well in advance of a worm's release, it is obvious that preventative measures just aren't working. New approaches to slowing the spread of worms need to be investigated, and the TARPIT option for IPtables firewalls is one such option. Thanks to Nicolas Lidzborski for his assistance with this research.

Author Info

[Tony Bautts](#) is a security consultant who has written and contributed to three security books from Syngress Publishing. He is currently working on a fourth, "The Linux Network Administrator's Guide 3rd Edition" for O'Reilly and Associates. In addition to writing, Tony has spoken at several national information security conferences.

[Privacy Statement](#)

Copyright 2006, SecurityFocus