the Scanner Tarpit HOWTO

John D. Hardin <*jhardin@impsec.org*>

$Revision: 0.15 $ $Date: 2006-12-15 21:03:16-08 $

How to configure a Linux firewall protecting a publicly-accessible (boundary, DMZ) network to detect worms' and attackers' scanning activity and react in real time to block and interfere with that scanning activity. A discussion of reporting tools and possible extensions is also included, with details for setting up an SMTP-only tarpit.

Contents

1 Introduction

1.1 Introduction

Typically the first step in attacking a remote network is to scan it for vulnerabilities. A vulnerability might be a computer running a service *daemon* that has a *Buffer Overflow* bug that can be exploited to run arbitrary commands at an elevated privilege level. Once the “vulnerability footprint” of a network is known, the attacker can focus attention on the vulnerable computer(s), gain entry, and use the compromised computer to attack other computers.

In addition to a system cracker intentionally focusing an attack on a particular network, or simply scanning to see if a vulnerability exists, there are programs (worms) that use these same vulnerabilities to spread autonomously, scanning for vulnerable systems and automatically infecting any that are discovered. The Microsoft IIS “Code Red” and “Nimda” worms are particularly high-profile examples of worms that continue to be active even today.

Also, many publicly-accessible FTP servers are misconfigured to allow anonymous write access. Many people actively scan for FTP servers seeking misconfigured servers they can use to exchange stolen software (“ *warez* ”) and entertainment media (e.g. movies) for free.

This document describes how to configure a Linux firewall that is guarding a publicly accessible network (e.g. a “boundary network” or “DMZ”) so that the firewall will (1) automatically detect the initial scan for vulnerabilities and (2) react in real time to protect the network. In addition to simply blocking the host that is performing the scan, the firewall will (3) act to interfere with the scan, both to interfere with the attacker and to slow the spread of autonomous worms, and (4) report the scanning activity to the appropriate responsible parties.

This document also briefly covers using "tarpit" software to respond to or prevent specific types of abuse, with a focus on email.

1.2 Audience

This document’s intended audience is Linux network administrators. You are assumed to have familiarity with the concepts of TCP/IP networking, firewall design principles, and with the syntax of the firewall commands in Linux. In the examples we will use the syntax as supported by the 2.2 kernel; the commands should be easy to translate to 2.4 and subsequent kernels.

You may have to rebuild your firewall kernel to enable support for Transparent Proxy, so you should be familiar with and comfortable with configuring, building, and installing customized kernels.

It is possible that these tools and techniques can be used on other Unix-flavor operating systems. If you are able to implement a scan-trap on another OS and it is similar to what is outlined here, please let me know; I'd like to include instructions for that as well.

1.3 Tools

Our scanner tarpit will be built by tying together a suite of powerful and freely-available tools and services. These include:

Abacus Portsentry
`http://freshmeat.net/projects/portsentry/`

LaBrea
`http://sourceforge.net/projects/labrea`

LogSentry (formerly LogCheck)
`http://freshmeat.net/projects/logsentry/`

DShield
`http://www.dshield.org/`

MRTG
`http://freshmeat.net/projects/mrtg/`

Note that we're not going to be using an intrusion-detection tool such as *Snort* - this document assumes you don't care to determine exactly what vulnerability the attacker or worm is trying to exploit. In fact, tarpitting the scanning traffic will interfere with the ability of an IDS tool to analyze that traffic, since it's being throttled down to nothing.

1.4 Feedback, Credits & Resources

The home page for the Scanner Tarpit HOWTO is `http://www.impsec.org/linux/security/scanner-tarpit.html`

Please feel free to send any feedback or comments regarding this document to me at <*jhardin@impsec.org*> . The current version can be found at:

- HTML: `http://www.impsec.org/linux/security/scanner-tarpit.html`
- Postscript: `http://www.impsec.org/linux/security/scanner-tarpit.ps.gz`
- PDF: `http://www.impsec.org/linux/security/scanner-tarpit.pdf`
- RTF: `http://www.impsec.org/linux/security/scanner-tarpit.rtf`
- ASCII text: `http://www.impsec.org/linux/security/scanner-tarpit.txt`

- SGML source: `http://www.impsec.org/linux/security/scanner-tarpit.sgml`

Thanks to Tom Liston for thinking up and coding LaBrea.

Thanks to Eric Raymond for maintaining *the Jargon File* , and Denis Howe for *The Free On-line Dictionary of Computing* .

The author's home page is at `http://www.impsec.org/~jhardin/`

1.5 Copyright & Disclaimer

2 Background Knowledge

2.1 What is a firewall?

A ***firewall*** is, in its simplest form, a device that connects two networks and controls the traffic that may pass between those networks. Generally a firewall is used to make sure outsiders can only access those resources the administrator wishes to make available to the public.

The most common way to do this is through packet filters. A packet filter identifies network traffic by specifying its source and destination using the *IP addresses* and possibly *port numbers* . Once traffic has been identified it can be permitted or blocked, or subjected to further processing.

Some resources for more information on firewalls:

- Network Computing's *Network Design Manual* at `http://www.networkcomputing.com/netdesign/wall2.html`
- *O'Reilly & Associates'* book Building Internet Firewalls , ISBN 1-56592-871-7.
- the Linux Documentation Project *IPChains HOWTO*

2.2 What is a Boundary Network or DMZ?

A common network configuration is to have *two* firewalls: one between the public Internet and a small "boundary network" or "Demilitarized Zone" (DMZ), where publicly-visible computers (e.g. a public HTTP and FTP server, the public mail server) are attached. Then a second firewall is placed between the boundary network and the private network.

This way both the public and the users on the private network have access to the servers on the boundary network, but an attacker on the Internet would have to penetrate two firewalls to gain access to the private network. This allows much more control over access, and separating the security between the public and the boundary network from the security between the public and the private network makes it easier to design and audit the firewall system as a whole. It also spreads the packet filtering load over two machines.

2.3 What is a firewall script?

A **firewall script** is a shell script that contains a series of firewall commands that configure the operating system's packet filters. Typically this script is run whenever the firewall system is rebooted.

Firewall scripts usually define *environment variables* to improve their readability. The firewall script examples in this document will do this as well.

Here are the firewall environment variables used in our examples:

- `$ANY` specifies "any host" - `0.0.0.0/0`
- `$INET_IF` specifies the Internet network interface - e.g. `eth1` for DSL or `wp1_chdlc` for a T1.

These variables would be set at the top of your firewall script.

2.4 What is a tarpit?

A ***tar pit*** is a big pit with tar in it. Large animals would become mired in the tar and eventually die through starvation or suffocation. When applied to a network firewall, it means much the same thing.

The *TCP communication protocol* provides *flow control* settings which allow the communicating computers to set the size of data packets being exchanged, and ensures reliable communications through retransmission of packets whose receipt has not been acknowleged.

A TCP tarpit is a program that sets the flow control settings to, essentially, *prevent* communication rather than facilitate it. It sets the packet data and "unacknowleged data" (window) size parameters to very low values (e.g. the protocol overhead plus only one data byte), which slows the transmission rate to a trickle. Then it never acknowledges packets, so transmission of that one byte will be retried over and over, ideally bringing the transmitting program (the scanning tool or worm) to a virtual halt for several hours.

More information is available on the LaBrea website at

`http://www.hackbusters.net/LaBrea/` - this site is required reading.

2.5 What is Transparent Proxy?

Transparent Proxy is a facility provided by the Linux kernel. It provides the capability of taking traffic that would normally pass *through* a gateway and redirecting it instead to the gateway itself.

The most common application for this is transparent proxy of HTTP traffic. The firewall redirects traffic destined for port 80 on some *other computer* to the local port on the firewall that the proxy server is listening on. This way web browsers on the local network can browse the Internet via the proxy, *without* being explicitly told to use the proxy.

We will use this facility to enable us to detect scans of the entire boundary network, not just scans of the outer firewall itself.

More information (focusing on the use of transparent proxy in the above HTTP setting) is available on the LinuxDoc website at

`http://www.linuxdoc.org/HOWTO/mini/TransparentProxy.html`

3 Planning

The implementation of a scan-trap depends on two assumptions:

First, that there are certain types of traffic that should *never* be coming in from the Internet - for example, SNMP, NFS (and related RPC protocols), and LPD. Many “back door” remote access *trojans* tend to listen on specific port numbers as well. Traffic destined to these services can, with a large degree of safety, be assumed to be hostile in all cases.

Note that given the propensity of unsophisticated Windows users to misconfigure their computers, inbound NetBIOS-over-TCP traffic should *not* be considered hostile - but don’t forget to block it at the firewall!

Second, that most traffic destined to *nonexistent hosts* - IP addresses on the boundary network that are not in use - is the result of random scanning or scanning of entire blocks of network addresses. Such traffic should not, however, trigger an instantaneous response. Human error in misconfiguring an application or typing errors in URLs can generate traffic to nonexistent network addresses, and traffic to network addresses that were used in the past can persist long after the network address is no longer in use.

For purposes of illustration our boundary network will be 192.168.0.x

3.1 Traffic that is always hostile

So, the first step is to generate a list of service requests that should be considered hostile whenever received from the Internet. Here is a starting point for TCP and UDP port numbers:

```
98         LinuxConf (system configuration, not from the Internet!)
111        Sun RPC (crackers scanning for Buffer Overflow flaws
             permitting remote access, typically as root)
161        SNMP (BO flaws, insecure default configurations)
162        SNMP (BO flaws, insecure default configurations)
512        rexec (weak authentication)
513        rlogin (cracking weak passwords)
514        rsh (weak authentication)
```

515	LPD (BO flaws)
635	NFS mounts
1433	Microsoft SQL server (default install has blank admin password)
9704	various Windows back doors...
12345	
12346	
16959	
22222	
27374	
31337	
32771	
32772	
32773	
32774	
33270	
34555	
35555	
54320	
54321	...my, there are a lot of these... :)

To keep this list up-to-date, you should monitor resources such as the SANS alert mailing list at `http://www.sans.org/newlook/digests/SAC.htm` , the CERT advisory mailing list at `http://www.cert.org/contact_cert/certmaillist.html` , and (of course) the BugTraq mailing list.

3.2 Traffic to nonexistent hosts

And the second step is to generate a list of service requests and unused boundary network addresses (or netblocks) that can be considered indications of hostile activity when taken together.

For the purposes of illustration we'll say that there are two large unused blocks of addresses on our boundary network: 192.168.0.128 - 192.168.0.159 and 192.168.0.192 - 192.168.0.223. If you get out your IP calculators, you'll see that those netblocks are 192.168.0.128/27 and 192.168.0.192/27. We'll assume that these addresses have never been used for any purpose that the public would know about.

For those netblocks we don't need to repeat the above hostile service list. Instead, we need to define services that are subject to attack but are more legitimately expected to be coming in from the Internet because there may indeed be such servers available to the public somewhere on the boundary network:

21	FTP (warez kiddies scanning for anonymous-writable servers)
22	SSH (BO flaws)
23	TELNET (BO flaws, weak passwords)
25	SMTP (spammers scanning for open relays)
53	DNS server (BO flaws)
80	HTTP (to catch HTTP worms scanning for vulnerable servers)
1080	SOCKS proxy (identity hiding, bounce attacks on others)
1723	PPTP server (DoS bugs, weak passwords)
3128	Squid HTTP/FTP proxy (DoS bugs, identity hiding)
3389	Microsoft Terminal Server (DoS bugs, weak passwords)
8080	HTTP proxy (identity hiding)

This traffic cannot, by itself, be considered hostile. To be considered hostile, it also needs to be destined to a computer that does not exist.

4 Detecting and blocking scans of the boundary network

A great tool for detecting scanning activity is *Abacus Portsentry* . You configure it to listen for network traffic on certain ports, and if a given host generates a goodly amount of traffic to those ports in a short period of time, a warning is generated and action is taken. It is intended to detect and react to the second stage of an attack scan: scanning all of the ports on a computer to see what services it is running, in an attempt to detect vulnerable services or determine what operating system is in use.

Though Portsentry is powerful, it is also simple (which is good), and is only designed to protect a single host. We can, however, use transparent proxy to enable Portsentry to detect scans of the *entire boundary network*. Using transparent proxy we will redirect scanning traffic from the hosts on the boundary network to the firewall itself, where all of the scanning traffic will be received by Portsentry.

4.1 Configuring Portsentry

To configure Portsentry to detect scans, edit the Portsentry configuration file (e.g. `/etc/portsentry/portsentry.conf`) and set the `TCP_PORTS` and `UDP_PORTS` lists to the “always hostile” port list you’ve compiled.

Transparent proxy is configured via firewall rules. For example, if we wished the gateway to receive all of the HTTP (80/tcp) traffic passing through it, we could run a firewall command similar to this:

```
/sbin/ipchains -A input -i $IFACE -p tcp -d $ANY 80 -j REDIRECT 8080
```

Meaning: any traffic coming in on interface `$IFACE`, protocol TCP, destined to port 80 on any machine anywhere, will be redirected to port 8080 on the local host. This illustrates the original goal of Transparent Proxy: proxying HTTP traffic without having to explicitly configure each individual local network client to use the proxy.

To use transparent proxy to allow PortSentry to trap scans of the entire boundary network, we must run a series of commands like:

```
/sbin/ipchains -A input -i $INET_IF -d $ANY 111 -j REDIRECT 111
```

Thus, any traffic coming into the outer firewall that is destined for port 111 on any host on the boundary network (whether or not it exists) will be redirected to port 111 on the firewall itself, where Portsentry is listening.

And:

```
/sbin/ipchains -A input -i $INET_IF -d 192.168.0.128/27 21 -j REDIRECT 12345
```

Thus, any traffic coming into the outer firewall that is destined for port 21 on any host in an unused address range on the boundary network will be redirected to port 12345 on the firewall itself, where Portsentry is listening.

The traffic destined for unused ranges on the boundary network must be redirected to some port on the firewall where portsentry is listening. This may not be the same port that the traffic was originally destined to, if there are legitimate servers for those ports elsewhere on the boundary network.

This is best done in a couple of loops near the top of your firewall script, after your private-IP and antispoofing rules:

```
# flytrap for port scanners and other d00ds - entire boundary network
for port in 98 111 161 162 512 513 514 515 1433 9704 12345 12346 16959 22222 27374 31337 32771 32772 32773 327
do
    /sbin/ipchains -A input  -j REDIRECT $port -l -i $INET_IF -p tcp -d $ANY $port
    /sbin/ipchains -A output -j DENY              -i $INET_IF -p tcp -s $ANY $port
    /sbin/ipchains -A input  -j REDIRECT $port -l -i $INET_IF -p udp -d $ANY $port
    /sbin/ipchains -A output -j DENY              -i $INET_IF -p udp -s $ANY $port
done

# trap scans on unused blocks of our Class-C
for port in 21 22 23 25 53 80 1080 1723 3128 3389 8080
do
    /sbin/ipchains -A input  -j REDIRECT 12345  -l -i $INET_IP -p tcp -d 192.168.0.128/27 $port
    /sbin/ipchains -A output -j DENY               -i $INET_IP -p tcp -s 192.168.0.128/27 $port
    /sbin/ipchains -A input  -j REDIRECT 12345  -l -i $INET_IP -p tcp -d 192.168.0.192/27 $port
    /sbin/ipchains -A output -j DENY               -i $INET_IP -p tcp -s 192.168.0.192/27 $port
done
```

These scripts are, of course, subject to more refinement based on your needs and are only presented as an illustration.

Portsentry's reaction to a scan is configurable. For our purposes, we'll block the scanning host by running the following script.

`/etc/portsentry/portsentry.blacklist`:

```
ATTACKER=$1
# record the attacker's IP address
echo "${ATTACKER}/32" >> /etc/IP-BlackList
# to manage load, don't log non-SYN traffic
/sbin/ipchains -A i-i-blk -j DENY   -p tcp ! -y -s $ATTACKER
/sbin/ipchains -A i-i-blk -j DENY               -s $ATTACKER -l
# let tarpit ACKs back out, block the rest
/sbin/ipchains -A o-i-blk -j ACCEPT -p tcp      -d $ATTACKER
/sbin/ipchains -A o-i-blk -j DENY               -d $ATTACKER
```

Tell Portsentry to run this script in response to a scan by editing `/etc/portsentry/portsentry.conf` and setting the following parameter:

```
KILL_ROUTE="/etc/portsentry/portsentry.blacklist $TARGET$"
```

4.2 The `i-i-blk` and `o-i-blk` firewall chains

The i-i-blk (input-internet-block) and o-i-blk (output-internet-block) firewall chains used by Portsentry are built near the top of the main firewall script; we re-read `/etc/IP-Blacklist` so that a firewall restart won't suddenly let a scanner or worm back into the boundary network:

```
# The blacklist
if [ -s /etc/IP-BlackList ]
then
    /sbin/ipchains -N i-i-blk
    /sbin/ipchains -A input  -i $INET_IF -j i-i-blk
    /sbin/ipchains -N o-i-blk
    /sbin/ipchains -A output -i $INET_IF -j o-i-blk
    (
    /usr/bin/perl -n -e 's/#.*//;
      if (
           /([0-9]+\.[0-9]+\.[0-9]+\.[0-9]+\/[0-9]+)/
         ) {print "$1\n";}' /etc/IP-BlackList |\
    uniq | tail -200 | sort | uniq | \
    while read IP_TO_BLOCK
    do
        # to manage load, don't log non-SYN traffic
        /sbin/ipchains -A i-i-blk -j DENY   -p tcp ! -y -s $IP_TO_BLOCK
        /sbin/ipchains -A i-i-blk -j DENY               -s $IP_TO_BLOCK -l
        # let tarpit ACKs back out, block the rest
        /sbin/ipchains -A o-i-blk -j ACCEPT -p tcp      -d $IP_TO_BLOCK
        /sbin/ipchains -A o-i-blk -j DENY               -d $IP_TO_BLOCK
    done
    ) &
fi
# The flytrap code goes AFTER this point
```

This script only re-blocks up to 200 of the most-recently-seen scanning hosts. This is done to limit system load and to allow for scanners from dynamic IP addresses: we don't want to block or tarpit a dialup or DHCP IP address permanently. Also, even if the scanning traffic is coming from a static IP address, a compromised host being used for scanning may be resecured or the scanner may eventually be booted from their ISP if they are scanning from their own computer.

Note that the i-i-blk and o-i-blk chains must be built in your firewall script BEFORE the rules that redirect traffic to Portsentry, or the firewall will never actually block the attacking IP address.

The first part of our scan-trap is done. We can now detect scans on our boundary network and block the scanner in real time.

5 Tarpitting the scanner or worm

Blocking the scanner or worm protects *our* network from further attack. We'd also like to be good network citizens and make the scanner's life hard, to take some of the load off other admins who might be the attacker's next target.

The tool for doing this is called LaBrea. Basically it plays tricks with the TCP flow control settings to tell the scanning computer "Yes, I'm here, but I can only accept one byte per packet," and then doesn't acknowlege packets so the sender sits in a timeout-and-retransmit loop for long periods of time. Ideally the scanner will hit the tarpit and become stuck for hours on end, trying vainly to see whether we're running a vulnerable version of IIS and leaving some other unprotected network alone.

This response is effective where the scanner is checking for version banners or other indications of a specific vulnerability, or is a worm trying to propagate itself. On scans that are only looking for running servers (e.g. "half-open" or "stealth" scans) this won't slow the scanner, but has the benefit of polluting the scanner's database of "hits" - most of the hosts on the boundary network will be reported as running the service that the scanner is looking for. Even if the host does not really exist.

LaBrea is designed to be installed as the only service running on a dedicated "tarpit host" that is sitting on the boundary network. However, the fact that we are using transparent proxy and Portsentry to detect and block scans of the boundary network means that this model won't work. Instead, we'll run LaBrea on the outer firewall itself.

5.1 Running LaBrea on the outer firewall

(NOTE: a new LaBrea is currently in beta release, and some changes are needed for it to run on a T1 vs. on a standard Ethernet interface, and to improve its behavior in the application described here. See the patch file for the needed changes. They may already be in the beta when you read this.)

LaBrea has sophisticated facilities for acting as if it were all of the nonexistent hosts on the boundary network in a dynamic manner. Since we're running it on the outer firewall, and we only want to tarpit hosts after a scan is detected, we need to turn off some of the smarts in LaBrea, and tell it explicitly which hosts we want it to tarpit. Fortunately, LaBrea supports a packet filtering mechanism via `libpcap`. Make sure you have a recent version of `libpcap` - older versions have a bug that prevents proper operation when the filter passes a certain size.

To generate the packet filter we need to capture the attacking IP addresses in a file, which the Portsentry blacklist script is doing. This file is then used to generate the packet filter file for LaBrea.

`/usr/local/sbin/tarpit`:

```
#!/bin/bash
INET_IF=wp1_chdlc   # the T1 - may be eth1 for non-T1 firewalls
NETWORK=' -n 192.168.0.0 -m 255.255.255.0 '
EVIL=/etc/IP-BlackList
BPF=/usr/local/etc/LaBrea-filter

# don't run this script in parallel
# this could definitely use some improvement
while [ -s $BPF.lock ]
do
        if ps -p `cat $BPF.lock` >/dev/null 2>&1
        then
                # lock is valid
                if [ -s $BPF.lock.2 ]
                then
                        # another rebuild already queued
                        exit
                else
                        echo $$ > $BPF.lock.2
                        sleep 5
                        rm -f $BPF.lock.2
                fi
        else
                rm -f $BPF.lock $BPF.lock.2
        fi
done

echo $$ > $BPF.lock

if [ -s $EVIL ]
then
    rm -f $BPF
    echo 'tcp and src host ( ' > $BPF

    /usr/bin/perl -n -e 's/#.*//;
      if (
           /([0-9]+\.[0-9]+\.[0-9]+\.[0-9]+)\/[0-9]+/
         ) {print "$1\n";}' $EVIL |\
    uniq | tail -200 | sort | uniq | sed -e 's|/.*||' \
    while read IP_TO_BLOCK
    do
        echo "$IP_TO_BLOCK or " >> $BPF
    done

    echo '127.0.0.1 )' >> $BPF
fi

rm -f $BPF.lock $BPF.lock.2

if ps -ef | grep -v grep | grep -q Labrea
then
        # re-read packet filter file
        killall -HUP LaBrea
else
        # start LaBrea
        exec /usr/local/sbin/LaBrea -z -i $INET_IF -l -v -j -x $NETWORK -F $BPF
fi
```

To manage LaBrea automatically, we make a "respawn" entry in `/etc/inittab`:

```
# run the tarpit daemon
tp:345:respawn:/usr/local/sbin/tarpit
```

And in the `/etc/portsentry/portsentry.blacklist` script that Portsentry runs to blackhole a new scanning host, we append a command to run the tarpit script to update the packet filter:

```
nice /usr/local/sbin/tarpit
```

6 Reporting scanning activity

Blocking the scanner protects us, and tarpitting the scanner helps to protect others, but what really needs to happen is notification of a responsible party so that the scanning activity *stops*, either through cleaning and securing a compromised or infected host or having the scanner's account terminated and Internet access removed.

To do this, you can watch your logs and manually track down responsible parties and send them email when you're attacked. This is, however, a lot of work.

Fortunately there are resources to automate this process. We'll tie together two tools: LogCheck (now LogSentry) periodically processes new log entries, and DShield serves as a central clearing house for collecting attack data and notifying responsible parties. We'll use LogCheck to collect attack data from our logs, and add the capability to have it mail the relevant portions automatically to DShield.

6.1 LogCheck - notify yourself

forthcoming

6.2 DShield - notify the ISP and others

DShield monitors incoming data from many sites, and when it appears an attack is underway (e.g. enough suspicious traffic is coming from a given IP) then the person responsible for that IP will be determined and notified that a computer in their domain is attacking others.

The benefits to this are:

1. You don't have to track down the ISP and send the notification yourself.
2. Dshield collects information from a large number of sources. If the scanner's ISP receives a report that 15,000 hosts have been attacked then they may be more inclined to take action quickly than they would be if they were notified that 15 hosts have been attacked.
3. Dshield acts as a repository for attack data. The more sites that report attack data to Dshield, the more likely a nascent widespread attack (e.g. Code Red XVI) is to be detected quickly.

6.3 MRTG - graph the activity

(covers configuring SNMP to report tarpit traffic statistics, and configuring mrtg to log and report those statistics)

7 Protecting individual computers

The techniques described in this paper are focused on a firewall protecting a large public network. There's no reason these tools cannot be used to protect small networks, e.g. a home network behins a Linux firewall connected to the Internet via DSL.

A discussion of this is forthcoming...

8 Possible extensions

8.1 Spam control

The abuse of email by unsolicited bulk mailers continues to increase. One of the reasons it is so appealing to advertisers is that very large numbers of people can be contacted for a very small investment (by the spammer) in time and resources.

Setting up a tarpit for spam attacks both of these economies. If the spammer's bulk mail program comes to a halt when it tries to send messages to a particular mail server, the spammer's investment in time (the time to send an individual message and the total time to send a bulk mailing) and resources (the number of simultaneous messages in the process of being delivered and the system load that causes) go up. If many sites implement spam tarpits, the investment goes *way* up.

For the same reasons this can also be useful as negative feedback for the operator of an open SMTP relay. A small ISP operating an open relay may simply ignore messages that bounce from sites employing a RBL filter, particularly if those sites are not sites that the ISP or their legitimate customers regularly communicate with. The ISP may also not notice the extra load on their SMTP server caused by the relayed spam messages. If a spammer relays a large number of messages via their server, and all of the delivery threads on that server get tarpitted, then the ISP *will* notice - their regular mail delivery will be severely interfered with, and the costs of the spammer's trespass will be highlighted.

The ethical aspects of this response to spam deserve discussion. Is it a Denial-of-Service attack if *you* have not initiated the communication? Is it "polite" to cause a third-party SMTP server to completely bog down when they try to send mail to you? Is it ethical to actively entice spammers to add tarpitted email addresses to their mailing lists? The responses will, of course, cover the full spectrum based upon how rabidly spam is despised. For the purposes of discussion I'll offer four possible paths of action: passive, reactive, proactive and aggressive trapping.

Note: In the current version of LaBrea (2.5-stable-1) there is no provision for producing a welcome banner message. The prevents effective tarpitting of SMTP traffic since the SMTP procotol specifies the server must send a welcome banner before the client attempts to send any data. The client will hit the tarpit, not see a welcome banner, and time out after a brief delay.

Patches are available on SourceForge to add the ability to send a greeting banner so the mail client can be tarpitted when they attempt to send the HELO command. See

`http://sourceforge.net/tracker/index.php?func=detail&aid=1612818&group_id=70896&atid=529395`

(there are related bugfix and other capability patches referred to on that page as well, you want them all). If the current version is greater than 2.5-stable-1 these patches may already have been incorporated into the mainline version.

8.1.1 Passive

Set up SMTP tarpitting on unused IP addresses in your netblock. This will serve to trap people scanning for open relays.

8.1.2 Reactive

Set up a SMTP-only tarpit on your public mail server and only tarpit repeat abusers; for example, clients that have already been rejected more than twice by your MTA based on DNS RBLs you are using ("three strikes, you're out"), or clients that ignore the SMTP RFC by sending data before receiving the server's greeting banner (see "Greet_Pause").

A script that does this for sendmail on Linux is available at

`http://www.impsec.org/~jhardin/antispam/spammer-firewall`

- it requires the above patches to be applied to LaBrea. This script can be installed on the same system as your public sendmail MTA and it will only tarpit repeat offenders' SMTP traffic.

8.1.3 Proactive

The above script could be used as a basis for setting up a static proactive SMTP tarpit that firewalls and captures any clients that appear in various RBL lists (if the RBL data is available in CIDR format), rather than only tarpitting clients who have already attempted delivery and been rejected.

This requires a source for regular updates of the desired RBLs in CIDR format. Such a feed may require a contract and a monetary investment, and consumes greater resources on the tarpitting system than the Reactive method.

8.1.4 Aggressive

briefly:

Set up a SMTP tarpit on a machine that does not otherwise receive email. Configure LaBrea to accept SMTP traffic to that host from any other host (e.g. don't limit the tarpit clients by IP). Publish that machine as the MX host for a subdomain of your domain (e.g. `@tarpit.example.com`).

Then invite the spammers in.

Post to newsgroups that spammers troll using email addresses from that domain (e.g. `spam-me@tarpit.example.com`). Respond to spamlist "remove" addresses with your From: set to that address. Publish the email address in non-visible mailto: links on web pages. Publish the SMTP tarpit server in publicly-accessible open relay lists (which spammers no doubt use for their purposes). Etc.

This may get you sued. Implement at your own risk. I specifically disclaim any responsibility for any legal issues or punishment you may experience from doing this.