

NETWORK SECURITY HACKS™

100 Industrial-Strength Tips & Tools



O'REILLY®

Andrew Lockhart

HACK
#40

Block OS Fingerprinting

Keep outsiders on a need-to-know basis regarding your operating systems.

When performing network reconnaissance, one very valuable piece of information for would-be attackers is the operating system running on each system discovered in their scans. From an attacker's point of view, this is very helpful in figuring out what vulnerabilities the system might have or which exploits may work on a system. Combined with the knowledge of open ports found during a port-scan, this information can be devastating. After all, an RPC exploit for SPARC Solaris isn't very likely to work for x86 Linux—the code for the portmap daemon isn't common to both systems, and they have different processor architectures. Armed with the knowledge of a given server's platform, attackers can very efficiently try the techniques most likely to grant them further access without wasting time on exploits that cannot work.

Traditionally, individuals performing network reconnaissance would simply connect to any services detected by their port-scan, to see which operating system the remote system is running. This works because many daemons, such as Sendmail, Telnet, and even FTP, readily announce the underlying operating system, as well as their own version numbers. Even though this method is easy and straightforward, it is now seen as intrusive since it's easy to spot someone connecting in the system log files. Additionally, most services can be configured not to disclose this sensitive information. In response, more sophisticated methods were developed that do not require a full connection to the target system to determine which operating system it is running. These methods rely on the eccentricities of the host operating system's TCP/IP stack and its behavior when responding to certain types of packets. Since individual operating systems respond to these packets in a particular way, it is possible to make a very good guess at what OS a particular server is running based on how it responds to *probe packets*, which normally don't show up in log files. Luckily, such probe packets can be blocked at the firewall to circumvent any operating system detection attempts that deploy methods like this.

One popular tool that employs such OS detection methods is Nmap (<http://www.insecure.org/nmap/>), which not only allows you to detect the operating system running on a remote system, but also perform various types of port-scans.

Attempting to detect an operating system with Nmap is as simple as running it with the `-O` switch. Here are the results of scanning an OpenBSD 3.3 system:

```
# nmap -O pufffy
```

```
Starting nmap 3.48 ( http://www.insecure.org/nmap/ ) at 2003-12-02 19:14 MST  
Interesting ports on pufffy (192.168.0.42):
```

```
(The 1653 ports scanned but not shown below are in state: closed)
PORT    STATE SERVICE
13/tcp  open  daytime
22/tcp  open  ssh
37/tcp  open  time
113/tcp open  auth
Device type: general purpose
Running: OpenBSD 3.X
OS details: OpenBSD 3.0 or 3.3
```

```
Nmap run completed -- 1 IP address (1 host up) scanned in 24.873 seconds
```

To thwart Nmap's efforts, we can employ firewall rules that block packets used for operating-system probes. These are fairly easy to spot, since several of them have invalid combinations of TCP flags. Some of the tests that Nmap performs cannot be blocked by PF by simply adding block rules, but they can be blocked if stateful filtering and a default deny policy have been implemented in the ruleset. This is because some of the tests make use of TCP options, which cannot be filtered with PF.

To block these fingerprinting attempts with OpenBSD's PF, we can put rules similar to these in our */etc/pf.conf*:

```
set block-policy return

block in log quick proto tcp flags FUP/WEUAPRSF
block in log quick proto tcp flags WEUAPRSF/WEUAPRSF
block in log quick proto tcp flags SRAFU/WEUAPRSF
block in log quick proto tcp flags /WEUAPRSF
block in log quick proto tcp flags SR/SR
block in log quick proto tcp flags SF/SF
```

This also has the side effect of logging any attempts to the *pflog0* interface. Even if we can't block all of Nmap's tests, we can at least log some of the more unique attempts, and possibly confuse it by providing an incomplete picture of our operating system's TCP stack behavior. Packets that have triggered these rules can be viewed with *tcpdump* by running the following commands:

```
# ifconfig pflog0 up
# tcpdump -n -i pflog0
```

Now let's look at the results of an Nmap scan after enabling these rules:

```
# nmap -O puffy

Starting nmap 3.48 ( http://www.insecure.org/nmap/ ) at 2003-12-02 22:56 MST
Interesting ports on puffy (192.168.0.42):
(The 1653 ports scanned but not shown below are in state: closed)
PORT    STATE SERVICE
13/tcp  open  daytime
22/tcp  open  ssh
37/tcp  open  time
```

Block OS Fingerprinting

```
113/tcp open  auth
No exact OS matches for host (If you know what OS is running on it, see
http://www.insecure.org/cgi-bin/nmap-submit.cgi).
TCP/IP fingerprint:
SInfo(V=3.48%P=i686-pc-linux-gnu%D=12/2%Time=3FCD7B3F%O=13%C=1)
TSeq(Class=TR%IPID=RD%TS=2HZ)
T1(Resp=Y%DF=Y%W=403D%ACK=S+++%Flags=AS%Ops=MNWNNT)
T2(Resp=Y%DF=Y%W=0%ACK=S%Flags=AR%Ops=)
T3(Resp=Y%DF=Y%W=0%ACK=0%Flags=AR%Ops=)
T4(Resp=Y%DF=Y%W=4000%ACK=0%Flags=R%Ops=)
T5(Resp=Y%DF=Y%W=0%ACK=S+++%Flags=AR%Ops=)
T6(Resp=Y%DF=Y%W=0%ACK=0%Flags=R%Ops=)
T7(Resp=Y%DF=Y%W=0%ACK=S+++%Flags=AR%Ops=)
PU(Resp=Y%DF=N%TOS=0%IPLen=38%RIPTL=134%RID=E%RIPCK=F%UCK=E%ULEN=134%DAT=E)
```

Nmap run completed -- 1 IP address (1 host up) scanned in 27.028 seconds

As you can see, this time the attempt was unsuccessful. But if you are feeling particularly devious, simply confusing Nmap attempts may not be enough. What if you want to actually trick would-be attackers into believing that a server is running a different OS entirely? For example, this could be useful when setting up a [honeypot \[Hack #94\]](#) to attract miscreants away from your critical servers. If this sounds like fun to you, read on.