

Know Your Enemy: II

Tracking the blackhat's moves

Honeynet Project

<http://project.honeynet.org>

Last Modified: June 18, 2001

This article is the second of a series of articles. In the first article, [Know Your Enemy](#), we covered the tools and methodologies of the Script Kiddie. Specifically, how they probe for vulnerabilities and then attack. The [third paper](#) covers what script kiddies do once they gain root. Specifically, how they cover their tracks and what they do next. This, the second paper, will cover how to track their movements. Just as in the military, you want to track the bad guys and know what they are doing. We will cover what you can, and cannot determine, with your system logs. You may be able to determine if you are being probed, what you were being probed for, what tools were used, and if they successful. The examples provided here focus on Linux, but can apply to almost any flavor of Unix. Keep in mind, there is no guaranteed way to track the enemy's every step. However, this article is a good place to start.

Securing Your Logs

This article is not on Intrusion Detection, there are a variety of excellent sources that cover IDS. If you are interested in intrusion detection, I recommend checking out applications such as [Network Flight Recorder](#) or [snort](#). This article focuses on intelligence gathering. Specifically, how to figure out what the enemy is doing by reviewing your system logs. You will be surprised how much information you will find in your own log files. However, before we can talk about reviewing your logs, we first have to discuss securing your system logs. Your log files are worthless if you cannot trust the integrity of them. The first thing most black-hats do is alter log files on a compromised system. There are a variety of rootkits that will wipe out their presence from log files (such as cloak), or alter logging all together (such as trojaned syslogd binaries). So, the first step to reviewing your logs is securing your logs.

This means you will need to use a remote log server. Regardless of how secure your system is, you cannot trust your logs on a compromised system. If nothing else, the black-hat can simply do a `rm -rf /*` on your system, wiping your hard drive clean. This makes recovering your logs somewhat difficult. To protect against this, you will want all your systems to log traffic both locally and to a remote log server. I recommend making your log server a dedicated system, ie. the only thing it should be doing is collecting logs from other systems.. If money is an issue, you can easily build a linux box to act as your log server. This server should be highly secured, with all services shut off, allowing only console access (see [Armoring Linux](#) for an example). Also, ensure that port 514 UDP is blocked or firewalled at your Internet connection. This protects your log server from receiving bad or un-authorized logging information from the Internet.

For those of you who like to get sneaky, something I like to do is recompile syslogd to read a different configuration file, such as `/var/tmp/.conf`. This way the black-hat does not realize where the real configuration file is. This is simply done by changing the entry `"/etc/syslog.conf"` in the source code to whatever file you want. We then setup our new configuration file to log both locally and to the remote log

server ([see example](#)). Make sure you maintain a standard copy of the configuration file, `/etc/syslog.conf`, which points to all local logging. Even though this configuration file is now useless, this will throw off the black-hat from realizing the true destination of our remote logging. Another option for your systems is to use a secure method of logging. One option is to replace your `syslogd` binary with something that has integrity checking and a greater breadth of options. One option is `syslog-ng`, which you can find at <http://www.balabit.hu/products/syslog-ng.html>

Most of the logs we will use are the ones stored on the remote log server. As mentioned earlier, we can be fairly confident of the integrity of these logs since they are on a remote and secured system. Also, since all systems are logging to a single source, it is much easier to identify patterns in these logs. We can quickly review what's happening to all the systems in one source. The only time you would want to review logs stored locally on a system is to compare them to what the log server has. You can determine if the local logs have been altered by comparing them to the remote logs.

Pattern Matching

By looking at your log entries, you can usually determine if you are being port scanned. Most Script Kiddies scan a network for a single vulnerability. If your logs show most of your systems being connected from the same remote system, on the same port, this is most likely an exploit scan. Basically, the enemy has an exploit for a single vulnerability, and they are scanning your network for it. When they find it, they exploit it. For most Linux systems, TCP Wrappers is installed by default. So, we would find most of these connections in `/var/log/secure`. For other flavors of Unix, we can log all `inetd` connections by launching `inetd` with the `-t` flag, facility daemon. A typical exploit scan would look like something below. Here we have a source scanning for the `wu-ftpd` vulnerability.

```
/var/log/secure
```

```
Apr 10 13:43:48 mozart in.ftpd[6613]: connect from 192.168.11.200
Apr 10 13:43:51 bach in.ftpd[6613]: connect from 192.168.11.200
Apr 10 13:43:54 hadyen in.ftpd[6613]: connect from 192.168.11.200
Apr 10 13:43:57 vivaldi in.ftpd[6613]: connect from 192.168.11.200
Apr 10 13:43:58 brahms in.ftpd[6613]: connect from 192.168.11.200
```

Here we see the source `192.168.11.200` scanning our network. Notice how the source sequentially scans each IP (this is not always the case). This is the advantage of having a log server, you can more easily identify patterns in your network since all the logs are combined. The repeated connections to port 21, ftp, indicated they were most likely looking for the `wu-ftpd` exploit. We have just determined what the black-hat is looking for. Often, scans tend to come in phases. Someone will release code for an imap exploit, you will suddenly see a rush of imaps scans in your logs. The next month you will be hit by ftp. An excellent source for current exploits is <http://www.cert.org/advisories/> Sometimes, tools will scan for a variety of exploits at the same time, so you may see a single source connecting to several ports.

Keep in mind, if you are not logging the service, you will not know if you are scanned for it. For example, most `rpc` connections are not logged. However, many services can simply be added to `/etc/inetd.conf` for logging with TCP Wrappers. For example, you can add an entry in `/etc/inetd.conf` for NetBus. You can define TCP Wrappers to safely deny and log the connections (see [Intrusion Detection](#) for more info on this).

What's the Tool?

Sometimes you can actually determine the tools being used to scan your network. Some of the more basic tools scan for a specific exploit, such as [ftp-scan.c](#). If only a single port or vulnerability is being probed on your network, they are most likely using one of these "single mission" tools. However, there exist tools that probe for a variety of vulnerabilities or weaknesses, the two very popular tools are sscan by jsbach and [nmap](#) by Fyodor. We selected these two tools because they represent the two "categories" of scanning tools. We highly recommend you run these tools against your own network, you may be surprised by the results :)

NOTE: The tool sscan is severely out of date. sscan is discussed only as an example. For scanning your own network for vulnerabilities, we highly recommend the OpenSource tool [Nessus](#).

sscan represents the "all purpose" Script Kiddie scanning tool. It probes a network for a set of specific vulnerabilities. It is customizable, allowing you to add probes for new exploits. You just give the tool a network and network mask, and it does the rest for you. However, the user must be root to use it. The output is extremely easy to interpret (hence making it so popular): It gives a concise summary of many vulnerable services. All you have to do is run sscan against a network, grep for the word "VULN" in the output, and then run the "exploit du jour". Below is an example of sscan ran against the system mozart (172.17.6.30).

```
otto #./sscan -o 172.17.6.30
```

```
-----<[ * report for host mozart *
<[ tcp port: 80 (http) ]>           <[ tcp port: 23 (telnet) ]>
<[ tcp port: 143 (imap) ]>         <[ tcp port: 110 (pop-3) ]>
<[ tcp port: 111 (sunrpc) ]>       <[ tcp port: 79 (finger) ]>
<[ tcp port: 53 (domain) ]>       <[ tcp port: 25 (smtp) ]>
<[ tcp port: 21 (ftp) ]>

--<[ *OS*: mozart: os detected: redhat linux 5.1
mozart: VULN: linux box vulnerable to named overflow.
<[ *CGI*: 172.17.6.30: tried to redirect a /cgi-bin/phf request.
<[ *FINGER*: mozart: root: account exists.
<[ *VULN*: mozart: sendmail will 'expn' accounts for us
<[ *VULN*: mozart: linux bind/iquery remote buffer overflow
<[ *VULN*: mozart: linux mouted remote buffer overflow
-----<[ * scan of mozart completed *
```

Nmap represents the "raw data" tool set. It doesn't tell you what vulnerabilities exist, rather, it tells you what ports are open, you determine the security impact. Nmap has quickly become the port scanner of choice, and with good reason. It takes the best of a variety of port scanners and puts all their functionality into a single tool, including OS detection, various packet assembly options, both UDP and TCP scanning, randomization, etc. However, you need networking skills to use the tool and interpret the data. Below is an example of nmap ran against the same system.

```
otto #nmap -sS -O 172.17.6.30
```

```
Starting nmap V. 2.08 by Fyodor (fyodor@dhp.com, www.insecure.org/nmap/)
```

```
Interesting ports on mozart (172.17.6.30):
```

Port	State	Protocol	Service
21	open	tcp	ftp
23	open	tcp	telnet
25	open	tcp	smtp
37	open	tcp	time
53	open	tcp	domain
70	open	tcp	gopher
79	open	tcp	finger
80	open	tcp	http
109	open	tcp	pop-2
110	open	tcp	pop-3
111	open	tcp	sunrpc
143	open	tcp	imap2
513	open	tcp	login
514	open	tcp	shell
635	open	tcp	unknown
2049	open	tcp	nfs

```
TCP Sequence Prediction: Class=truly random
```

```
Difficulty=9999999 (Good luck!)
```

```
Remote operating system guess: Linux 2.0.35-36
```

```
Nmap run completed -- 1 IP address (1 host up) scanned in 2 seconds
```

By reviewing your logs, you can determine which of these tools were used against you. To do this, you have to understand how the tools work. First, an sscan will log in as follows (this is a default scan with no modifications to any config files):

```
/var/log/secure
```

```
Apr 14 19:18:56 mozart in.telnetd[11634]: connect from 192.168.11.200
Apr 14 19:18:56 mozart imapd[11635]: connect from 192.168.11.200
Apr 14 19:18:56 mozart in.fingerd[11637]: connect from 192.168.11.200
Apr 14 19:18:56 mozart ipop3d[11638]: connect from 192.168.11.200
Apr 14 19:18:56 mozart in.telnetd[11639]: connect from 192.168.11.200
Apr 14 19:18:56 mozart in.ftpd[11640]: connect from 192.168.11.200
Apr 14 19:19:03 mozart ipop3d[11642]: connect from 192.168.11.200
Apr 14 19:19:03 mozart imapd[11643]: connect from 192.168.11.200
Apr 14 19:19:04 mozart in.fingerd[11646]: connect from 192.168.11.200
Apr 14 19:19:05 mozart in.fingerd[11648]: connect from 192.168.11.200
```

/var/log/maillog

```
Apr 14 21:01:58 mozart imapd[11667]: command stream end of file, while
reading line user=??? host=[192.168.11.200]
Apr 14 21:01:58 mozart ipop3d[11668]: No such file or directory while reading
line user=??? host=[192.168.11.200]
Apr 14 21:02:05 mozart sendmail[11675]: NOQUEUE: [192.168.11.200]: expn root
```

/var/log/messages

```
Apr 14 21:03:09 mozart telnetd[11682]: ttloop: peer died: Invalid or
incomplete multibyte or wide character
Apr 14 21:03:12 mozart ftpd[11688]: FTP session closed
```

sscan also scans for cgi-bin vulnerabilities. These probes will not be logged by syslogd, you will find them in access_log. I decided to included them anyway for your edification :)

/var/log/httpd/access_log

```
192.168.11.200 - - [14/Apr/1999:16:44:49 -0500] "GET /cgi-bin/phf
HTTP/1.0" 302 192
192.168.11.200 - - [14/Apr/1999:16:44:49 -0500] "GET /cgi-bin/Count.cgi
HTTP/1.0" 404 170
192.168.11.200 - - [14/Apr/1999:16:44:49 -0500] "GET /cgi-bin/test-cgi
HTTP/1.0" 404 169
192.168.11.200 - - [14/Apr/1999:16:44:49 -0500] "GET /cgi-bin/php.cgi
HTTP/1.0" 404 168
192.168.11.200 - - [14/Apr/1999:16:44:49 -0500] "GET /cgi-bin/handler
HTTP/1.0" 404 168
192.168.11.200 - - [14/Apr/1999:16:44:49 -0500] "GET /cgi-bin/webgais
HTTP/1.0" 404 168
192.168.11.200 - - [14/Apr/1999:16:44:49 -0500] "GET /cgi-bin/
websendmail HTTP/1.0" 404 172
192.168.11.200 - - [14/Apr/1999:16:44:49 -0500] "GET /cgi-bin/webdist.
cgi HTTP/1.0" 404 172
192.168.11.200 - - [14/Apr/1999:16:44:49 -0500] "GET /cgi-bin/faxsurvey
HTTP/1.0" 404 170
192.168.11.200 - - [14/Apr/1999:16:44:49 -0500] "GET /cgi-bin/
htmlscript HTTP/1.0" 404 171
192.168.11.200 - - [14/Apr/1999:16:44:49 -0500] "GET /cgi-bin/pfdisplay.
cgi HTTP/1.0" 404 174
192.168.11.200 - - [14/Apr/1999:16:44:49 -0500] "GET /cgi-bin/perl.exe
HTTP/1.0" 404 169
192.168.11.200 - - [14/Apr/1999:16:44:49 -0500] "GET /cgi-bin/wwwboard.
pl HTTP/1.0" 404 172
192.168.11.200 - - [14/Apr/1999:16:44:50 -0500] "GET /cgi-bin/ews/ews/
```

```

architext_query.pl HTTP/1.0" 404 187
192.168.11.200 - - [14/Apr/1999:16:44:50 -0500] "GET /cgi-bin/jj
HTTP/1.0" 404 163

```

Notice how a complete connection was made for all the ports(SYN, SYN-ACK, ACK) then torn down. That is because sscan is determining at the application layer what is going on. Not only does sscan want to know **if** your ftp port is open, but **what** ftp daemon is running. The same can be said for imap, pop, etc. This can be seen in sniff traces using sniffit, a tool commonly used to sniff passwords. <

```

mozart $ cat 172.17.6.30.21-192.168.11.200.7238
220 mozart.example.net FTP server (Version wu-2.4.2-academ[BETA-17])(1) Tue
Jun 9 10:43:14 EDT 1998) ready.

```

As you see above, a complete connection was made to determine the version of wu-ftpd that was running. When you see the complete connections in your logs, as shown above, you are most likely being scanned by an exploit tool. These tools are making a complete connection to determine what you are running.

Nmap, like most port scanners, does not care **what** you are running, but **if** you are running specific services. For this, nmap has a powerful set of options, letting you determine what kind of connection to make, including SYN, FIN, Xmas, Null, etc. For a detailed description of these options, check out http://www.insecure.org/nmap/nmap_doc.html. Because of these options, your logs will be different based on the options selected by the remote user. A connection made with the -sT flag is a complete connection, so the logs will look similar to sscan, however by default nmap scans more ports.

```

/var/log/secure
Apr 14 19:18:56 mozart in.telnetd[11634]: connect from 192.168.11.200
Apr 14 19:18:56 mozart imapd[11635]: connect from 192.168.11.200
Apr 14 19:18:56 mozart in.fingerd[11637]: connect from 192.168.11.200
Apr 14 19:18:56 mozart ipop3d[11638]: connect from 192.168.11.200
Apr 14 19:18:56 mozart in.telnetd[11639]: connect from 192.168.11.200
Apr 14 19:18:56 mozart in.ftpd[11640]: connect from 192.168.11.200
Apr 14 19:19:03 mozart ipop3d[11642]: connect from 192.168.11.200
Apr 14 19:19:03 mozart imapd[11643]: connect from 192.168.11.200
Apr 14 19:19:04 mozart in.fingerd[11646]: connect from 192.168.11.200
Apr 14 19:19:05 mozart in.fingerd[11648]: connect from 192.168.11.200

```

One thing to keep in mind is the -D (or decoy) option. This nmap option allows the user to spoof the source address. You may see scans from 15 different sources at the same time, but only one of them is the real one. It is extremely difficult to determine which of the 15 was the actual source. More often, users will select the -sS flag for port scanning. This is a stealthier option, as only a SYN packet is sent. If the remote system responds, the connection is immediately torn down with a RST.

```

/var/log/secure

```

Dooh! Notice there is nothing there! That is because system logs only record full connections. Since nmap scanner is using -sS option, a complete TCP connection is not being made, thus the scan attempts are not being logged. That is why this scanning method is considered stealthy, system logs are not recording the scans. For older Linux kernels, specifically 2.0.x, incomplete TCP connections are being logged, but as broken. The logs from a -sS scan looks as follows for older kernels. (**NOTE:** Only the first three entries are included here).

```
/var/log/secure
Apr 14 21:25:08 mozart in.rshd[11717]: warning: can't get client address:
Connection reset by peer
Apr 14 21:25:08 mozart in.rshd[11717]: connect from unknown
Apr 14 21:25:09 mozart in.timed[11718]: warning: can't get client address:
Connection reset by peer
Apr 14 21:25:09 mozart in.timed[11718]: connect from unknown
Apr 14 21:25:09 mozart imapd[11719]: warning: can't get client address:
Connection reset by peer
Apr 14 21:25:09 mozart imapd[11719]: connect from unknown
```

Notice all the errors in the connections. Since the SYN-ACK sequence is torn down before a complete connection can be made, the daemon cannot determine the source system. The logs show that you have been scanned, unfortunately you do not know by whom. This behaviour only happens with older 2.0.x linux kernels. To quote Fyodor " ... based on all the 'connection reset by peer' messages. This is a Linux 2.0.XX oddity -- virtually every other system (including the 2.2 and later kernels) will show nothing. That bug (accept() returning before completion of the 3-way handshake) was fixed."

Nmap includes other stealth option, such as -sF, -sX, -sN where various flags are used, This is what the logs look like for these scans

```
/var/log/secure
```

Once again, notice something here, no logs! Scary huh, you just got scanned and didn't even know it. All three types of scans determined the same results without them being logged, similar to the -sS option. To detect these stealth scans, you will need to use a different logging application such as [tcplogd](#) or [ippl](#). Most firewalls will also detect and log all of these scans, such as IPFilter, SunScreen, or FireWall-1.

Did They Gain Access?

Once you have determined that you were scanned, and what you were looking for, the next big question is "Did they get in?". Most of today's remote exploits are based on buffer overflows (otherwise known as smashing the stack). Simply stated, a buffer overflow is when a program (usually a daemon) receives more input than it expected, thus overwriting critical areas in memory. Certain code is then executed, usually giving the user root access. For more info on buffer overflows, check Aleph1's excellent paper [Smashing the Stack for Fun and Profit](#).

You can normally identify buffer overflow attacks in the /var/log/messages log file (or /var/adm/messages for other flavors of Unix) for attacks such as mountd. You will also see similar logs in maillog for such

Korn shell script

Conclusion

Your system logs can tell you a great deal about the enemy. However, the first step is guaranteeing the integrity of your log files. One of the best ways to do that is use a remote log server that receives and stores logs from all systems. Once secured, you can then identify patterns in your log files. Based on these patterns and log entries, you can determine what the black-hat is looking for, and potentially what tools they are using. Based on this knowledge, you can better secure and protect your systems.

The HoneyNet Project