



Global Information Assurance Certification Paper

Copyright SANS Institute
Author Retains Full Rights

This paper is taken from the GIAC directory of certified professionals. Reposting is not permitted without express written permission.

Interested in learning more?

Check out the list of upcoming events offering
"Hacker Tools, Techniques, and Incident Handling (Security 504)"
at <http://www.giac.org/registration/gcih>

Exploit Details

Name

The exploit is a buffer overflow exploit using cookies as the delivery mechanism.

Variants

This exploit is a proof of concept and as such it does not have any known variants. However, there are a vast number of buffer overflow exploits available.

The following are just a few examples:

- Vanity.c exploits the BNC IRC proxy.
- rlogin-exploit.c takes advantage of gethostbyname() in the rlogin routine of Solaris 2.5/2.5.1 to perform a root compromise.
- Sendmail, POP and IMAP contains so many buffer overflows that they are listed on the SANS Top 10 Internet Security Threats list.

Operating Systems

All operating systems

Protocols / Services

CGI

HTTP State Management Mechanism.(RFC 2109)

Brief Description

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.

CGI Protocol Description

The Common Gateway Interface protocol is a standard that allows a website user to communicate with programs running on the website's servers. A CGI program is essentially a program that the web server allows anyone in the world to run. Unlike a static web 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 4 ways: environment variables, the command line, standard input and standard output.

John M. Millican

SANS 2000 - San Jose

Incident Handling and Hacker Exploits Practical

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 will be 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 will be 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 website’s operations and possibly a foothold into the website’s network.

Cookie Protocol Description

John M. Millican

SANS 2000 - San Jose

Incident Handling and Hacker Exploits Practical

Cookies are a simple text-based mechanism to maintain state between websites and the clients that visit them. The HTTP protocol that websites 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 website 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 to a site, the processing and storage requirements to store the data at the website would be prohibitive.

In order to provide a sense of session or state to websites while minimizing the burden on the website, Netscape developed the specification for state objects, or cookies, to store the data on the client side and.

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 website by preceding the data to be stored with a Set-Cookie header that instructs the browser to store the data on the client system.

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

In this way state is maintained between the website 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 domain. An optional data item designates whether a secure connection is required. All subsequent data pairs are at the discretion of the website.

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 that websites may be collecting personal information or tracking the movements of their visitors across the Web. This perception is aggravated by market

John M. Millican

SANS 2000 - San Jose

Incident Handling and Hacker Exploits Practical

data collection companies such as DoubleClick that work in conjunction with websites for just that purpose.

The primary weaknesses are:

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

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. It is a classic case of trying to stuff a 5 pound casing with 10 pounds of meat more commonly known as a buffer overflow.

Buffer overflows work by violating how a computer processes program instructions and data in its memory. To better understand this we need to tell a story.

The Caterer of Buffer

There was once a caterer who lived in the peaceful and well-mannered country of Buffer. This caterer was famous throughout the land for his pancakes. Whenever someone in Buffer wanted to throw a gala party, they always had it at the banquet hall of the Caterer Of Buffer.

The caterer had a clever way of serving all the parties that were occurring within his banquet hall. He would take all the pancakes that were made for all the parties and put them in one big stack in the center of the banquet hall. Then whenever a guest from one of the parties wanted a pancake, he would serve them from the stack.

Now you can imagine how confusing it could be for the caterer to determine how many pancakes he would need for all the parties going on. But, he was a clever caterer, and he had a clever process to keep everything under control. Whenever he booked a party, he would ask how many pancakes the hostess wanted him to serve to her guests. On the night of the party, he would make however many pancakes had been requested and put them all on the stack.

Of course he had to keep track of which group of pancakes belonged to which party. He also had to keep track of how many pancakes each party had requested and already consumed. But, he had a simple solution to this challenge too.

John M. Millican
SANS 2000 - San Jose
Incident Handling and Hacker Exploits Practical

He found that if he kept a list of the top pancake for each party and if he only served the top pancake to its respective party, then all he would have to do was make sure no one from one party was served the top pancake from another party. This worked fine as long as no party ate more than its fair share. Naturally, well-mannered guests would never think of doing such a thing.

Then one day it happened. Travelers from the evil country of Hackdom came into the tranquil land of Buffer. The citizens of Buffer, being the peaceful and well-mannered people that they were, greeted them warmly and offered to honor the Hackers with a party. Wanting to impress the Hackers, they chose to have the party at the Caterer of Buffer's banquet hall.

Now we all know that Hackers can be such boorish pigs, and this was no exception. Instead of being polite, when know one was looking the greedy Hackers ate more than they should. After eating all the pancakes for their party, they started eating pancakes for all the other parties.

Imagine the anger of the caterer when he saw what those pigs had done! The caterer went ballistic, and he threw the bums out of his establishment! And, because the caterer was so angry, he also told the hostess in charge that she could never have another party at his banquet hall again.

The moral of the story, dear friends, is that if you want to maintain peace in the land of Buffer make sure your guests mind their manners and never eat more than their fair share.

The End

Now let's take our little story and rewrite it in computer terms.

The Operating System Of Computerland

There once was an operating system that lived in a computer. The operating system was famous for its memory segments. Whenever someone wanted to execute a program, they would ask the operating system to serve them memory segments in its computer.

The operating system served its memory segments from a central pool of memory. Managing all these memory segments for all the executing programs was tricky. To solve this problem the operating system arranged the memory into one big stack. Since the computer can execute several programs at once and each program can execute many functions, managing all the memory segments was a challenge.

But the operating system knew how to manage its resources. Whenever a program wanted to use some memory segments, the operating system would ask it how many segments were needed. It would then allocate that number of memory segments on the stack to the executing program. The operating system would also keep track of the boundaries of each group of memory segments. Whenever memory segments were used

John M. Millican

SANS 2000 - San Jose

Incident Handling and Hacker Exploits Practical

by a program, the operating system would take the segments from on top of the allocated portion of the stack. As long as the executing programs did not use more memory segments than they requested thus going across a memory boundary, all was well.

However, one day some clever hackers decided to see what would happen if they consumed more memory than they were allocated. When the operating system discovered what had been done, it core dumped and threw the hackers out. But because the hackers had also corrupted the program, the operating system threw it out too causing a denial of service to all of the program's other well-mannered functions.

The End

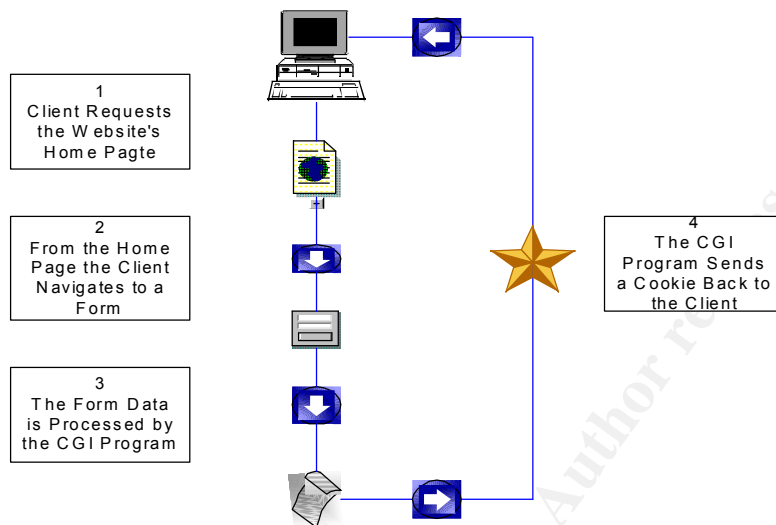
In a nutshell these fairy tales describe what occurs in a buffer overflow exploit, and that is the intent of this exploit – to cause a denial of service by overflowing the buffer of a vulnerable CGI application.

Why It Works

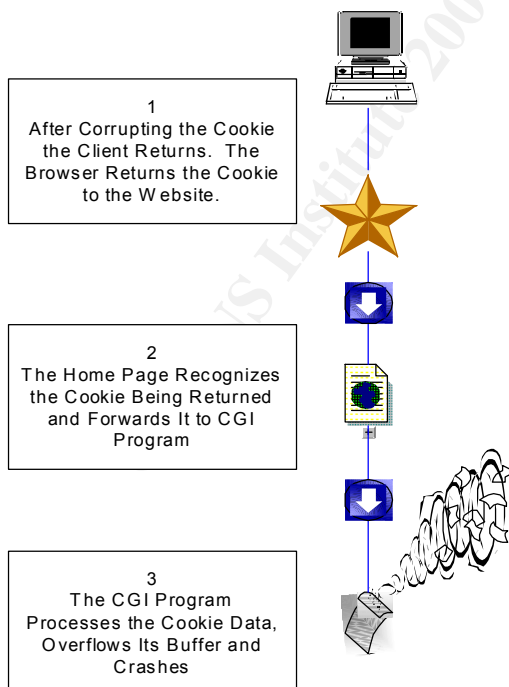
It works because the buffer overflow corrupts the server's memory stack. This corruption causes the program to crash. Skillfully designed buffer overflow exploits can be written in such a way as to allow the hacker to execute arbitrary commands with the privileges of the owner of the web programs.

Diagram of the Exploit

Initial Visit By The Client



Client Returns To Execute The Exploit



How To Use It

Included with this paper is a very simple website that demonstrates how cookies can be used to cause a denial of service attack. A more skilled programmer than me may be able to expand upon this and change it into a root exploit.

The essentials of this demonstration are:

1. The visitor is greeted by a registration page that invites them to input their name and address to personalize the site for them. If the visitor registers then the next time they visit the site they will be able to login to a personalized Welcome page that greets them by name.
2. The registration page processes the name and returns it to the visitor's browser in the form of a cookie for storage on the visitor's system.
3. The visitor returns to the website. Since a cookie exists for the website, the browser includes the cookie's data within the URL.
4. The website recognizes that cookie data has been returned with the visitor, and submits it to the CGI program to format a personalized greeting page.
5. The personalized welcome page is sent back to the visitor.

To perform the exploit all that needs to be done is to edit the cookie's content with your favorite text editor before returning to the website. In this demonstration, you would edit the cookie between steps 2 and 3 above.

To keep the programs simple (and therefore, within my limited programming skills), I did not have a CGI program running constantly in the background. Instead it is executed on demand. To better view the program crash and how it could cause a denial of service in a daemonized program, perform the following steps:

1. Perform steps 1 through 5 above.
2. Locate the cookie created by the website. The location varies by operating system and browser used.
3. Using a text editor, change one of the cookie values so that the combined length of the name is greater than 50 characters.
4. Revisit the site.
5. Choose the login link.
6. You should get an error message to the effect of "500 Internal Server Error".

If you are using Internet Explorer 5.0, you may have to take a different approach to see the error. IE 5.0 seems to recognize that the cookie has been tampered with and eats it. Therefore when the GET request is sent to the website, no cookie data is sent with it. To get around this, use a utility like nc to generate the GET request and append the cookie data to the end of the request. I did not have access to other browsers to test whether they acted the same way as IE.

John M. Millican
SANS 2000 - San Jose
Incident Handling and Hacker Exploits Practical

Signature of the Attack

How to Detect

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 that 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. Since NOP's 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 NOP's will overwrite the return address of the calling function. The command the hacker hopes to execute will follow the NOP's.

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 NOP's and the arbitrary command. After the function completes, it returns to the stack address that it was called from. It is the hacker's hope that he has overwritten that return address with the NOP's and that the system will execute them (i.e. do nothing) until it reaches the instructions he injected with the NOP's that are then executed.

How to Block

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

Patches Available

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 developer has published.

Procedural Solutions

The best protection against buffer exploit attacks is good programming techniques. Whereas you cannot eliminate the pipeline that they flow in, 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.

John M. Millican
SANS 2000 - San Jose
Incident Handling and Hacker Exploits Practical

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()`
- input while loops that do not explicitly check for overflows

Technical Solutions

While 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.

Since 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 could be used to provide this protection are encryption and MD5 checksums. By encrypting the data, the contents of the cookie would be 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 would be unencrypted, a new MD5 checksum would be 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" );

John M. Millican

SANS 2000 - San Jose

Incident Handling and Hacker Exploits Practical

```

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
#
# 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
```

John M. Millican
SANS 2000 - San Jose
Incident Handling and Hacker Exploits Practical
As part of GIAC practical repository.

```
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 );
}
```

Additional Information

Due to my limited programming skills, I used an object file to set the environment variables for the data sent to the CGI routines. These files are required to compile the above routines, but since the assignment instructions prohibited macros I did not include them. These files 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

© SANS Institute 2000 - 2002, Author retains full rights.