Global Information Assurance Certification Paper

Dennis Beach
Advanced Incident Handling Practical
Sans DC 2000

Exploit Details

Name: CGI-World Poll It Internal Variable Override Vulnerability
Variants: None Known
Operating System: All platforms running Perl
Protocols/Services: TCP/IP, HTTP, CGI
Brief Description:
A programming error allows to incoming variables to overwrite internal program variables. Input values can be passed which allow the client to view any file on the server for which the web server has read permissions.

Protocol Description

Hypertext Transfer Protocol (HTTP) operates over TCP/IP connections, usually to port 80, although other ports are frequently used. Many HTTP servers may operate on the same computer using different port numbers. The protocol is used to transfer documents between web servers and clients.

To conduct an HTTP session, the client first initiates a connection to the server. Once connected, the client sends a request message to the server, the server responds and closes the connection. HTTP is a stateless protocol meaning that each request requires a new connection, and the server does not keep track of you. The large majority of HTTP connections require no authentication making it an anonymous protocol as well.

HTTP 1.1 supports persistent connections which allow a client to make multiple requests over the same connection without being disconnected. This optional feature reduces the overhead of establishing connections. This feature does not maintain any state between the client and server beyond the existence of a connection.

In HTTP 1.0, there are only three kinds of requests:

1. GET - returns whatever information is requested.
2. HEAD - returns only the server's header information of the requested document.
3. POST - used for sending HTML form entries. This is the only request that sends a body with the request.

HTTP 1.1 supports additional request types.

An HTTP session can be manually executed over Telnet:

- First the telnet client connects to the server.

- Once connected, the client sends the request GET /index.html "HTTP/1.0" and enters a blank line to end the request.
- The server responds with a series of headers followed a single blank line, then the content requested. Then the session is terminated.

Transcript of HTTP Telnet session

```
> telnet www.lines.com 80
Trying 209.249.146.102...
Connected to www.lines.com.
Escape character is '^]'.
GET /index.html HTTP/1.0

HTTP/1.1 200 OK
Date: Wed, 16 Aug 2000 03:32:35 GMT
Server: Apache/1.3.9 (Unix)
Connection: close
Content-Type: text/html

<HTML>
<HEAD>
<TITLE>INFINITE LINES</TITLE>
</HEAD>
 ...
more body content
 ...
Connection closed by foreign host.
```

The Common Gateway Interface (CGI) is a standard for interfacing external applications with HTTP servers. CGI provides a means for an executable to run and have its output sent back to the requesting client.

A plain HTML document is static. A CGI program can be executed for each request, so it can output dynamic information. This provides for interactive sessions allowing searches, databases interactions, session tracking, state maintenance and more.

The CGI/1.1 Specification defines 4 methods of communication between the web server and the external program:

- Environment variables:
  Environment variables are used to send parameters to your program. the 2 most important are QUERY_STRING and PATH_INFO.

  QUERY_STRING contains the contents of the request string following the first ? . For the request http://someserver.com/cgi/some.cgi?a=3&b=fred the QUERY_STRING would contain a=3&b=fred.

  The arguments following the ? can either be hardcoded in a link or the results of an HTML form submission with the method of GET.

Additional information can be embedded in the path portion of the request. This extra information is made available in the PATH_INFO variable.

PATH_INFO is frequently used to indicate the location of files or file paths to the CGI program. The program can determine the location of the document relative to the DocumentRoot via the PATH_INFO environment variable.

An example which uses both environment variables is server-side image map processing. The request will be in the form
http://www.someserver.com/cgi/imagemap/path_to_mapfile/mapfile?x=3.4&y=4
5.
PATH_INFO will contain /path_to_mapfile/mapfile - the path and name of the mapfile
QUERY_STRING will contain the map coordinates(x,y) where the imagemap was clicked.

- The command line:
  The command line is only used for ISINDEX queries where the request has only a single argument in the form: /path_to_cgi/cgi?argument. The CGI is passed the single argument on the command line.
- Standard input:
  POST or PUT request sends its arguments as a block of data which is passed to the CGI program via standard in rather than in the QUERY_STRING.
- Standard output::
  The CGI program returns it's results to the web server via standard output.

The CGI program must parse the incoming data to extract the name/value pairs. For GET requests the QUERY_STRING will be in the form name=value&name=value. Form POST data comes in as a single block in the format from which the pairs must be extracted.

In both cases, the input may contain url-encoded characters. Because certain character have special meaning, ie, the character "&" separates name/value pairs, these characters are encoded using hex values. Additionally, spaces are treated as the end of a request string, so spaces are replaced with + signs. Within the CGI program, these values must be converted backe to their original ASCII representations, for example, a "/" is substitued for %20 in the incoming data.

In Perl this conversion can be accomplished by the following regular expressions:

```
$value = ~ s/ /+/g;    # Convert plusses back to spaces
$value =~ s/%([A-Fa-f0-9]{2})/pack("c",hex($1))/ge;    # Map hex encoded
characters back to ASCII
```

NOTE: Some exploits use url-encoding to disguise the actual data being sent to the CGI. This can allow the expoit code to escape detection by simple Intrusion Detection Systems and other text based filtering.

A CGI program can return many types of data, the most common is HTML, but it may return a text document or a generated image. So, the first line of the program's output must contain a header to tell the web server what type of data it is returning.

This header is passed to the client by the web server so the client knows how to handle the content, ie, display as HTML, launch a helper program, etc. The header is ASCII text and can span multiple lines. The entry is terminated by a single blank line. After which follows the content from the CGI program.

The CGI program can be written in nearly any language including Perl and C, as long as the program can conform to the CGI specification.

Description of variants

No variants of this attack are known. However, it is possible that other input variable based attacks would succeed due to the lack of input validation.

How the exploit works

Pollit is a Perl CGI script which provides online polls. The adminstrative function allows web-based poll creation and management. Visitors may vote on polls and see the real-time results. The poll questions and results are stored on the server in text files.

The program works by reading template files from its data directory, placing content in the templates if necessary, then returning the pages to the web server.

The exploit takes advantage of 3 problems in this program:

1. A programming error which allows the the overwriting of predefined variables
2. Lack of form input validation
3. The "Poison null-byte" vulnerability in Perl

Near the beginning of the script, the variable $data_dir is defined. This variable contains the path to the directory from which the program reads its data and template files. Because the program imports input variables directly into the main namespace, the incoming data_dir value replaces the predefined value.

The code excerpt below(sub ReadForm) shows portions the routine used by the program for handling incoming form values: the programming error is in the line $$name = $value. This causes GLOBAL variables to be assigned for each variable sent in the request. So the incoming data_dir value gets assigned to the GLOBAL $data_dir

overwriting the existing value. Note that there is no validation or cleaning of dangerous characters from incoming data. There is none anywhere else in the program.

```
==========================================================================

sub ReadForm {

  my($max) = $_[1];                                    # Max Input Size
  my($name,$value,$pair,@pairs,$buffer,%hash);                 # localize
variables

...

  @pairs = split(/&/, $buffer);                                # Split
into name/value pairs
  foreach $pair (@pairs) {

    ($name, $value) = split(/=/, $pair);               # split into $name
and $value
    $value =~ tr/+/ /;                                 # replace "+" with
" "
    $value =~ s/%([A-F0-9]{2})/pack("C", hex($1))/egi;       # replace
%hex with char

...
    $$name = $value;
    }

  return %hash;

  }
==========================================================================
```

Now that the $data_dir has been assigned to the value desired by the attacker, the program calls the Template subroutine to open the template:

&Template("$data_dir/_ssi_poll.html",'html');

The subroutine Template now attempts to open the file "/etc/passwd\0/_ssi_poll.html":

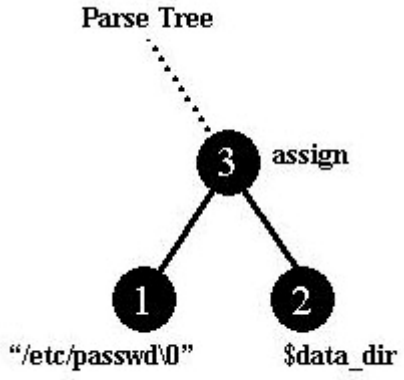
open(FILE, "<$_[0]");

The Perl runtime environment interprets this string as "/etc/passwd" truncating the string at the \0 (NULL). This is known as the "Poison NULL byte" vulnerability. The program then reads in the contents of the /etc/passwd file and sends the contents back to the browser as it would with any template file.

To understand the existence of the "Poison NULL byte" vulnerability, one must understand the mechanics of Perl script execution. When a Perl script is run, the Perl executable is called to first interpret, then execute that script. The program Perl itself is written in C.

The Perl executable parses the text of the script to create a plan for execution known as a "parse tree". The nodes on the parse tree are called opcodes, the smallest executable unit Perl deals with. These opcodes are then passed to the Perl runtime environment. The

runtime environment uses C functions to perform the necessary operations to implement the opcodes.



This figure shows the parse tree representation of assigning the exploit value into the $data_dir variable.

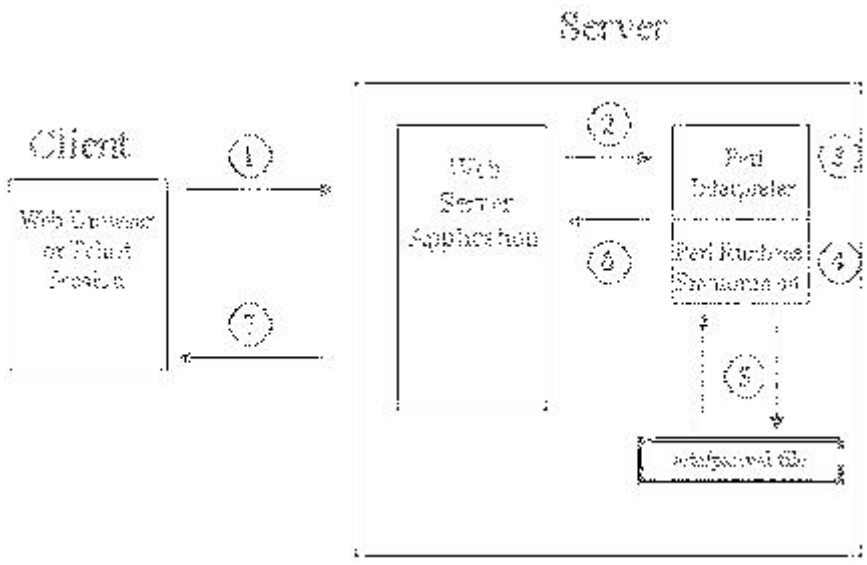
Each of the circles is a single opcode.

In the C programming language, a null byte (\0) signifies the end of a string, so any characters following the null are ignored. When the Poll It CGI is executed, the string "/etc/passwd\0/_ssi_poll.html" is sent to the Perl runtime environment to be opened by a C system level open command. The C function sees the string ending at "/etc/passwd" and opens that file.

Even on a well secured server, this exploit can cause great harm. The administrator password for Poll It is stored within the Poll_It_v2.0.cgi script. The web server must have read and execute permissions to run this script, so the exploit would allow an attacker to read that script.

With the administator password the attacker could alter the poll or the results data. Also, the script's administrator may use the same password in other places. The exploit would also lay open source code to other scripts and script data on the server, as the server must also have permissions for these files in order to run them. These may yield database passwords or passwords to connect to other machines.

Diagram of the exploit



1. The client makes an HTTP request:
   http://www.somehost.com/cgi-bin/pollit/Poll_It_SSI_v2.0.cgi?data_dir=/etc/passwd%00
2. The web server launches the Poll_It_SSI_v2.0.cgi Perl CGI process, passing the data_dir=/etc/passwd%00 via the QUERY_STRING environment variable
3. The Perl interpreter compiles the Poll_It_SSI_v2.0.cgi into opcodes
4. The Perl runtime environment is passed the string to open "/etc/passwd\0/_ssi_poll.html"
5. The program executes a system level open in C on "/etc/passwd" file and reads in the contents, because C sees the \0 null byte as the end of the string
6. The Perl CGI returns the contents of the file to the web server and terminates
7. The web server returns the file contents to the client

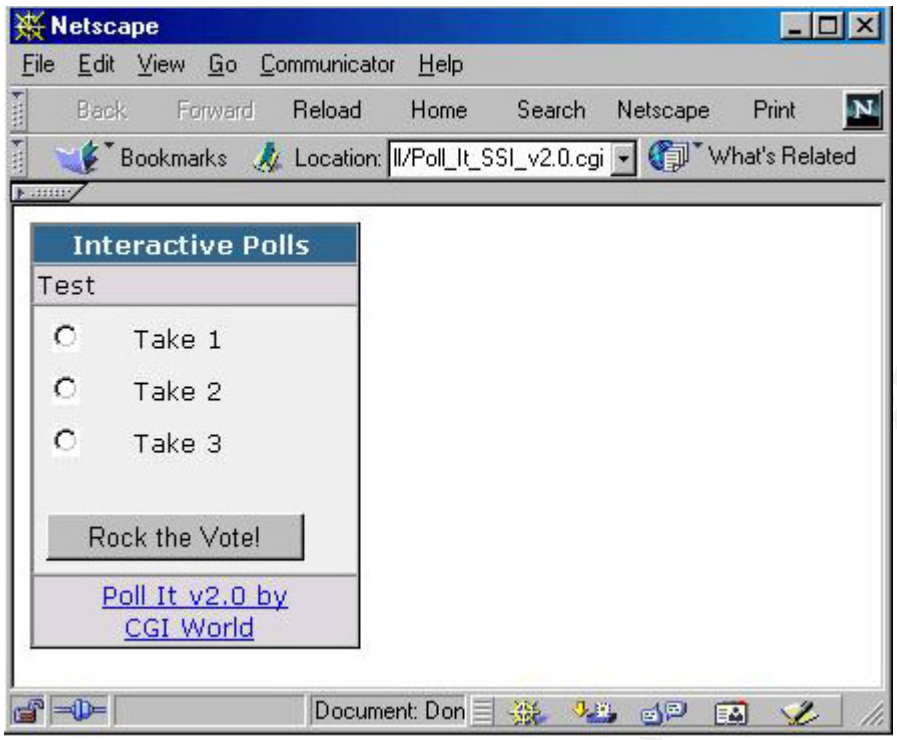
How to use the exploit

The exploit can implemented via any text or graphical web browser, or through telnet session to the HTTP port, typically port 80.

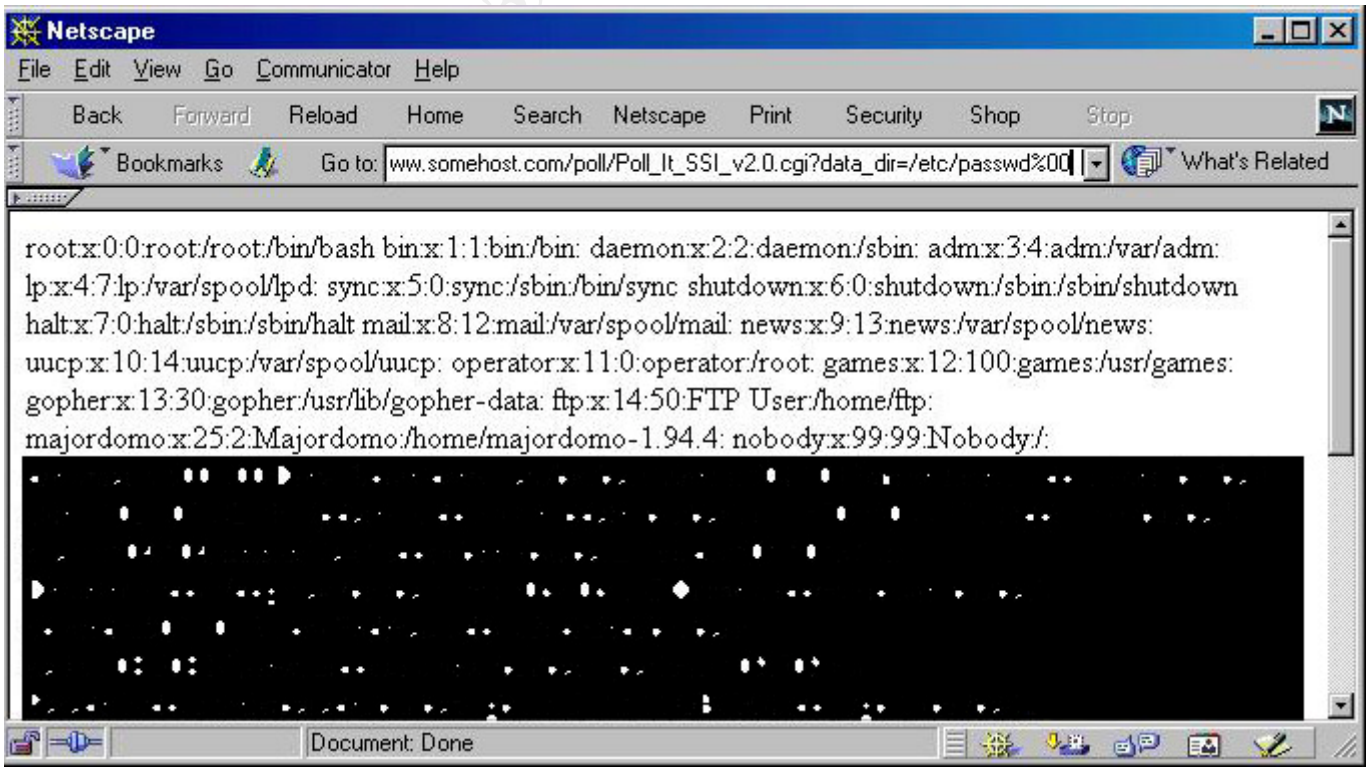
The basic format of the HTTP request(exploit) is:
http://servername/path_to_cgi?data_dir=desired_file%00
NOTE: Only files readable by the web server user or group can be viewed

To get the /etc/passwd file on the server www.somehost.com, the request string would be:
http://www.somehost.com/cgi-bin/pollit/Poll_It_SSI_v2.0.cgi?data_dir=/etc/passwd%00

The Poll It script was set up on a test server to illustrate the exploit.
This is the normal display page when the script is called with no arguments:
http://www.somehost.com/cgi-bin/pollit/Poll_It_SSI_v2.0.cgi



The next figure shows the display when script is called with the exploit request:
http://www.somehost.com/cgi-bin/pollit/Poll_It_SSI_v2.0.cgi?data_dir=/etc/passwd%00
(additional usernames have been blacked out)



Signature of the attack

Two items the same request string identify this attack:

1. The name of the cgi script: Poll_It_SSI_v2.0.cgi (unless it has been renamed)
2. %00 in request string or \0 after the query string has been unencoded

How to protect against it

The recommended patch is to move Line 78: %in = &ReadForm to line 66 before the internal variables are assigned. This solves the immediate problem of the internal variables being overwritten. However, this hack does not address the basic security issues with this program.

More secure protections:
1 - Scope Script Variables - Protect variables in the main namespace from collision with incoming variables or variables in other subroutines.

- Use CGI.pm: Perl 5 supports modules. Modules are code libraries which are invoked in a separate namespace. Variables in the main namespace can be explicitly referred to as main::$variable_name while variables in a module name other would be referred to as other::$variable_name.

  CGI.pm handles importing of incoming data placing name/value pairs in a separate namespace. The values can be safely used and examined without bringing them into the main namespace.

  Usage: use CGI.pm qw(standard);
  Input values can be accessed by: param(variable_name);

- Use strict: The strict pragma requires variables to be predeclared as global to the program or local to a subroutine. If a variable is not scoped, the program will give errors and exit without executing the script. This is a very helpful tool for any level programmer to ensure correct scoping.
- Use constant: Perl 5 supports constants by creating an anonymous subroutine that returns the value when called. The value the subroutine returns cannot be altered after compilation, so the constant is protected from being overwritten.

  Usage:
  use constant DATA_DIR => "/path/to/data/dir/";

  $a=DATA_DIR;
  $a now contains "/path/to/data/dir/"

2 - Validate form input - Form input should always be checked, any CGI program is potentially susceptible to problems from malformed input values, malicious or not.

- Use Taint: Taint mode can be used to enforce input validation. All data is initially considered tainted. The data must be untainted via a regular expression before it can be used in any operations. Taint can be invoked as a command line switch.

  Usage: #!/usr/bin/perl -T

  To untaint a variable it must be assigned after a regular expression check.
  For example:
  $a = param('a'); # $a is a tainted variable
  $a =~ /^(\d*)$/ ;# regular expression match for digits only
  $a = $1; # $a is now untainted, it has been assigned the result of the pattern match

- The null byte can be safely removed from any data by the following expression:
  $incoming_variable =~ s/\0//g;
- Only access expected variables
  If you know what variables should be coming in, look at those variables and leave the rest. One easy way to do this is to create an array of acceptable input variable names, then iterate over the array.

  Example:
  @accept_vars = qw(a b c d);
  foreach (@accept_vars) {
  $a = param($_);
  }

3 - Limit webserver privileges:

- Run the web server as a non-privileged user. This will limit the files the webserver is able to access. Do NOT run the web server as root.
- A more advanced solution would be to create a "Sandbox" for the CGI Scripts to run in. Chroot can been used to create a virtual root directory. The CGI application could not access above this root directory. See www.freebsd.org for more information.

Source code/ Pseudo code

Due to licensing restrictions, the source code is not included in this document, but can be downloaded from http://www.cgi-world.com/pollit.html

Additional Information

- Original BugTraq Mailing List Posting
- BugTrak Database Entry
- Discussion of Poison NULL byte and Metacharacters by Rain Forest Puppy
- Perl Home
- CGI.pm Homepage

- Perldoc Security Reference
- WWW Security FAQ
- CGI Specification
- Ch 18 Compiling
  Programming Perl, 3rd Edition
  Larry Wall, Tom Christiansen & Jon Orwant
  3rd Edition July 2000
  ISBN 0-596-00027-8