Writing secure CGI scripts

RUNNING A SECURE WEB SERVER

SIMSON L. GARFINKEL
simsong@vineyard.net

Web servers are designed to receive anonymous requests from unauthenticated hosts on the Internet and to deliver the requested information quickly and efficiently. As such, they provide a portal into your computer that can be used by friend and foe alike. ● No piece of software is without risk. Web servers, by their nature, are complicated programs. Furthermore, many organizations use Web servers with source code that is freely available over the Internet. Although this means the source code is available for inspection by the organization, it also means an attacker can scan it for vulnerabilities. ● The ability to add functions to a Web server using CGI scripts tremendously complicates security. While a CGI script can add new features to a Web server, it can also introduce security problems. For example, a Web server may be configured so that it can only access files stored in a particular directory on your computer, but a user may innocently install a CGI script that allows outsiders to read any file on your computer. Furthermore, because many users do not have experience in writing secure programs, locally written CGI scripts will likely contain bugs allowing an outsider to execute arbitrary commands on your system. Indeed, several books on CGI programming have included such flaws.

Because of the richness of its tools, the plethora of programming languages, and the ability of multiple users to be logged in at the same time from remote sites over a network, the UNIX operating system is a remarkably bad choice for running secure Web servers. Because many PC-based operating systems share many of these characteristics, they are also not very good choices. Experience has shown that the most secure Web server is a computer that:

- Runs only a Web server.
- Does not have a readily accessible scripting language.
- Does not support remote logins.

In practice, this describes an Apple Macintosh computer running MacHTTP, WebStar, or a similar Web server. According to recent surveys, such computers comprise as many as 15 percent of the Web servers on the Internet.

Of course, there are many advantages to running a Web server on a UNIX computer instead of a Macintosh. UNIX generally runs faster than Mac OS on comparable hardware, and is available for hardware platforms that run faster than PowerPC-based computers. Furthermore, it is generally easier for organizations to integrate UNIX-based Web servers with their existing information infrastructure, creating interesting possibilities for Web offerings. Finally, more MIS professionals are familiar with building UNIX-based Internet servers than Mac OS-based servers. Nonetheless, the security-conscious administrator should give the Mac-based approach serious thought.

To build a secure Web server on any platform, you must ensure that:

- Network users cannot execute arbitrary programs or shell commands on your server.
- CGI scripts that run on your server must either perform the expected function or return an error message. Scripts should expect and be able to handle any maliciously tailored input.
- If your server is compromised, an attacker will not be able to use it for further attacks against your organization.

Secure CGI Scripting

Writing a secure CGI script has all of the same problems as writing a secure SUID program or network server, plus many more. This is because the Web environment, the Web server, and the CGI script can interact in unexpected ways, creating problems with the combined system even though no problems were obviously present in any single part.

The more people writing scripts on your Web server, the greater the chance that one of those scripts will contain a significant flaw. Therefore, you should not allow users to place scripts on your server unless a qualified security professional has read through them and assured you of their safety.

One thing to watch for is trusting the user's browser. HTML has the ability to display selection lists, limit the length of fields to a certain number of characters, embed hidden data within forms, and specify variables that should be provided to CGI scripts. Despite all of this, your CGI script cannot depend on any of these restrictions because any CGI script can be run by directly requesting the script's URL; attackers do not need to go through your form or use the interface that you provide. So, be especially careful of the following:

- If you create a selection list, the value returned for the input field will not necessarily match the allowable values that you have defined.
- If you specify a maximum length for a variable, the length of the variable provided to your script may be significantly longer. (What would your script do if provided with a user name 4000 characters long?)
- Variables provided to your script may have names not defined in your script.
- The user may view data marked "hidden."

AVOIDING THE RISKS

The risks of eavesdropping affect all Internet protocols, but are of particular concern on the World Wide Web, where sensitive documents and other kinds of information—such as credit-card numbers—may be transmitted. Eavesdropping also makes possible traffic analysis, in which an attacker learns about the transactions performed by a target, without actually learning the content. The log files kept by Web servers are particularly vulnerable to this type of attack. To protect information from eavesdropping, you can either ensure that the information travels over a physically secure network (which the Internet is not) or encrypt the information so that it can only be decrypted by the intended recipient.

Eavesdropping Over the Wire

Information sent over the Internet must be encrypted to be protected from eavesdropping. Information sent over the Web can be encrypted in four ways:

Link encryption. The long-haul telephone lines used to carry the IP packets can be encrypted. Organizations can also use encrypting routers to automatically encrypt information destined for another office and sent over the Internet. Link encryption provides for the encryption off all traffic, but it can only be performed with prior arrangement. Link encryption has traditionally been very expensive, but new generations of routers and firewalls incorporate this feature at reasonably low cost.

Document encryption. Documents placed on the Web server can be encrypted with a system such as PGP (Pretty Good Privacy). Although this encryption provides for effective privacy protection, it is cumbersome because it requires the documents to be specially encrypted before they are placed on the server and specially decrypted when they are received.

Secure Socket Layer (SSL). SSL is a system designed by Netscape Communications that provides an encrypted TCP/IP pathway between two hosts on the Internet. SSL can be used to encrypt any TCP/IP protocol, such as HTTP, TELNET, or FTP, and can use a variety of public-key and token-based systems for exchanging a session key. Once a session key is exchanged, it can use a variety of secret-key algorithms. Currently, programs that use SSL and are distributed for anonymous FTP on the Internet or sold to international markets are restricted to a 40-bit key, which should not be consid-

- The values for variables provided may contain special characters.
- If you use cookies or special hidden tags to maintain state on your server, your script may receive cookies or tags that it never created.

Attackers are by definition malicious; they do not follow the rules. Never trust anything that is provided over the network.

When Testing is not Enough

Testing is performed on the contents of environment variables, host addresses, host names, URLs, user-supplied data, values chosen from selection lists, and even data that your script has inserted onto a WWW form using the hidden data type.

Consider the case of a CGI script that creates a series of log files for each host that contacts your WWW server. The name of the log file might be logfile/{hostname}. What will this program do if it is contacted by the host ../../../../etc/passwd.company.com? If improperly constructed, such a script could append a line to a system's /etc/passwd file. This could then be used to create unauthorized accounts on the system.

Beware of system(), popen(), pipes, backquotes, and Perl's eval() function.

Many programming languages—including C, ksh, sh, csh, and Perl—provide the means to spawn subprocesses, but you should avoid using these features when writing CGI scripts. When spawning a subprocess, don't pass through any strings that are provided by the user. If you must pass strings from the user to the subprocess, be sure not to pass shell metacharacters, such as the |, *, and & characters.

It is generally better to specify a list of allowable characters rather than one of dangerous characters. If you forget to specify an allowable character, no real harm is done; if you forget to specify a dangerous character, such as a backquote, you can compromise the security of your entire system.

Sending Mail

If you are writing a CGI script that allows a user to send mail, send it with the /usr/lib/sendmail program, rather than /bin/mailx or /usr/ucb/mail. The reason is that /usr/lib/sendmail does not have shell escapes, whereas the other mailers do.

Example 1 is a bit of Perl that you can use to send mail securely. It bypasses the shell by using exec () with a fixed string directly to run /usr/lib/sendmail.

Many Perl functions will run the shell, possibly without your knowledge. These include systemO, eval(), pipes, backquotes, and, occasionally, exec () (if shell metacharacters are present on the command line).

```
open (WRITE, "!-")
¦! exec ("usr/lib/sendmail", "-oi," "-t".)
¦! die "Can't fork$!\n":

print WRITE "To: $address\n";
print WRITE "Subject: $subject\n";
print WRITE "From: $sender\n";
print WRITE "\n$message\n.\n";
close(WRITE);
```

Example 1: Perl code to send mail securely.

Tainting with Perl

In Perl, you can use the "tainting" facility to track information provided by the user. Perl marks such information as "tainted." The only way to untaint information is to match it using a Perl regular expression, then copy out the matched values using Perl's string match variables.

For example, if a name has been provided by the user, you can untaint the value to be sure that it contains only letters, numbers, commas, spaces, and periods by using the Perl statements in Example 2.

There are two ways to enable tainting. In Perl 4, you should invoke the taintperl command by placing the following statement (or something similar) at the begin-

OF EAVESDROPPING

ered secure. A complete description of the SSL protocol can be found at http://home.netscape.com on the /newsref/std/SSL.html page.

SecureHTTP (SHTTP). SHTTP is an encryption system for HTTP designed by Commerce Net that only works with HTTP See http://www.eit.com/projects/s-http for details on the SHTTP protocol.

It is widely believed that most commercial Web servers that use encryption will use either SSL or SHTTP. Unlike the other aforementioned options, both SSL and SHTTP require special software both on the Web server and in the Web browser. This software will likely be in most commercial Web browsers in the future, although it may not be widespread in freely available browsers until the patents on public-key cryptography expire within the coming years.

When using an encrypted protocol, your security depends on several issues:

- The strength of the encryption algorithm.
- The length and secrecy of the encryption key.
- The reliability of the underlying software running on both the Web server and the Web client.

During the summer of 1995, a variety of articles was published describing failures of the encryption system used by the Netscape Navigator. In the first case, a researcher in France broke the encryption key used on a single message by using a network of workstations and two supercomputers. The message had been encrypted with the international version of Netscape Navigator using a 40-bit RC4 key. In the second case, a group of students at the University of California, Berkeley discovered a flaw in the random-number generator used by the UNIX-based version of Navigator. In the third case, the same group of students discovered a way to alter the binaries of the Navigator program as they traveled between the university's NFS server and the client workstation on which the program was actually running.

All of these attacks are highly sophisticated. Nevertheless, most of them do not seem to be having an impact on the commercial development of the Web. Within a few years, the encryption keys will hopefully be lengthened to 128 bits. Netscape has improved the process by which its Navigator program seeds its random-number generator. Surreptitiously altered binaries will likely not be a problem for most users with improved software-distribution systems based upon encryption and digital signatures.

```
$tainted_username =~ m/([a-zA-Z. ]*)/;
$untainted_username = $1;
You can use the following to extract an
                          e-mail address:
$tainted_email =~ /([\w-./]+\@[\w.-]+)/;
$untainted_email = $1;
```

Example 2: Perl statements to untaint a value.

ning of your file: `#!/usr/local/bin/taintperl`. If you are using Perl 5, you can accomplish the same result by using the -T flag: `#!/usr/local/bin/perl -T`.

Beware of Stray CGI Scripts

Most Web servers can be configured so that all CGI scripts are confined to a single directory (typically, cgi-bin). This configuration makes it easier for you to find and examine all of the CGI scripts on your system. Do not configure your Web server so that any file ending with the .cgi extension will run as a CGI script. Further more, you should:

- Use a program such as Tripwire to monitor for unauthorized changes to these scripts.
- Allow limited access to the script directory. Local users should not be allowed to install, remove, edit, or even read existing scripts without administrative review.
- Remove the backup files generated automatically by your editor. Many system administrators use text editors such as Emacs to edit scripts that are in place on the running server. Frequently, this leaves behind backup files with names such as "start-" or "create-account~," which can be executed by an attacker.

Keep Your Scripts Secret!

I deplore the practice of security through obscurity; that is, basing your system's security on its undocumented aspects. Nevertheless, the ready availability of UNIX source code—as opposed to the relative obscurity of the source code for other operating systems such as VMS or Windows/NT—allows potential attackers to search the operating system for avenues of attack, then design their strategy to guarantee the maximum possible access. One good way to prevent such attacks is to limit the access to source code.

Because it is so easy to make a mistake when writing a CGI program, CGI scripts and programs should be kept confidential. This does not guarantee security for buggy scripts: A determined attacker can still probe and frequently find flaws with your system. However, confidentiality significantly increases the amount of work required to find problems. Determined attackers will still get through, but casual ones may move on to other, more inviting systems.

One reason it is surprisingly easy to create an unsecure CGI script is the difficulty of testing your scripts against the various HTTP clients available. For example, most client programs will "escape," or specially encode, characters such as the backquote, which are specially interpreted by the UNIX shell. As a result, many CGI programmers do not expect the unencoded characters to be present in their applications' input streams, and they do not protect their scripts against that possibility. Characters in the input stream are either the result of a bug in a particular Web browser or, more likely, a malicious attempt to subvert your CGI script and gain control of your server.

Check *all* values provided to your program for special characters and appropriate length. The FTP service might be visible to someone using the WWW service, which would compromise its contents. Any real passwords in that file will be vulnerable.

Mixing HTTP with Anonymous FTP

Many sites store documents in the same directories that are accessed through anonymous FTP and the World Wide Web. For example, you may have a directory called /NetDocs on your server that is both the home directory of the FTP user and the root directory of your Web server. This would allow files to be referred to by two URLs, such as http://server.com/nosmis/myfile.html or ftp://server.com/nosmis/myfile.htrnl.

The primary advantages of HTTP over FTP are speed and efficiency. HTTP is optimized for anonymous access from a stateless server. FTP, on the other hand, had anonymous access added as an afterthought, and requires that the server maintain a significant amount of state for the client.

Nevertheless, many people prefer using FTP to transfer large files because the tools are different. Mixing HTTP and FTP directories poses a variety of security issues:

- Allowing anonymous FTP access to the HTTP directories allows users to bypass Web-server restrictions on document access.
- If potential attackers can FTP to your CGI scripts, they can search them for avenues for attack.
- If FTP directories are writable, an attacker may be able to upload scripts that will be run on your server.

Other Issues

You can take many other measures to secure your server. For example, you can limit the use of the computer so that it is solely a Web server. This will make it harder for an attacker to break in to your server and, if an attacker does, it will limit the amount of damage that can be done to the rest of your network.

If you make your server a standalone computer, isolate it from your network by:

- Deleting all unnecessary accounts.
- Not NFS mounting or exporting any directories.
- Deleting all compilers.
- Deleting all the utility programs that are not used during boot time or by the Web server.
- Providing as few network services as possible.
- Not running a mail server.

Another option—which may require a nontrivial amount of work—is to place your WWW server and all files in a separate directory structure. The WWW server is then wrapped with a small program that does a `chroot()` to the directory. Thus, if some way is found to break out of the controls you have placed on the server, the regular file system is hidden and protected from attack. Some WWW servers may have this approach included as an install-time option, so check the documentation.

Simson is coauthor of Practical UNIX and Internet Security *(O'Reilly & Associates, 1996), on which this article is based.*

URL RESOURCES

Lincoln Stein's WWW Security FAQ. **http://www-genome.wi.mit.edu/WWW/faqs/www-security-faq.html**

Paul Phillips' CGI security FAQ. **http://www.cerf.net/~paulp/cgi-security/**

NCSA's CGI security documentation. **http://hoohoo.ncsa.uiuc.edu/cgi/security.html**

The WWW section of the COAST Lab's hot list. **http://www.cs.purdue.edu/coast/hotlist.html**