Security considerations for Perl CGI Scripts

TS5004 Technical Communications

Katherine Rylien

rylien@yahoo.com

June 7, 2006

TABLE OF CONTENTS

ABSTRACT

CGI scripts written in Perl are very common on the internet, and are very useful for creating dynamic web pages. These pages can accept user input through forms and can modify a page based on the user's information, such as addressing the user by name, for example. But due to their complexity and the power of the language, many of these scripts contain security holes that can be exploited by hackers. This includes some very popular scripts that are available for download, free or otherwise. These are used by many web sites, both personal and commercial.

This paper discusses real-life problems caused by this issue, using specific examples of both malicious code and outcome. It then goes on to discuss possible solutions for the problem. These solutions include using scripts hosted on the server of a service provider, running scripts on ones own site that have been vetted by skilled programmers, and also what to watch for when writing scripts from scratch, modifying existing scripts or evaluating an unknown Perl script from a security standpoint. My intent is to make the paper accessible to an audience with a technical background in computers but not necessarily experienced in Perl programming.

INTRODUCTION

My first experience with writing Perl scripts was in a web development class I took during my final year of undergraduate studies. One person on my team, Dave Scott, knew a little about the language. I learned mostly by modifying his code as we created an online photo album program, and the following semester, I used what I had learned to create a dynamic list of web links for another class. The question of security never arose.

A couple of years later, I started to see some very strange behavior from a discussion board where I was very active. Dates and text appeared in the wrong locations, and then the site was taken down altogether. When it finally came back up, weeks later, the system administrator explained that she had been hacked. Further discussion revealed that the forum software was written in Perl—the language I remembered having so much fun with in school. I investigated the question of script security, and looking over a list of common programming errors that create security holes, I realized I was guilty of pretty much all of them.

THE PROBLEM

When a dynamic page allows users to enter data which is processed or stored on the server, it creates a vulnerability. For someone who wants to know how to exploit this, tips and tutorials can be found by doing a Google search for terms such as "hacking perl scripts" or "cgi vulnerability".

NMRC Example #1

One of my favorites is the Nomad Mobile Research Centre (NMRC), which offers a fairly comprehensive set of hacking instructions, dramatically displayed on a black background. To give just one example, NMRC provides instructions on how to exploit a test script installed by default on many servers. The intent of the script, test-cgi, is to confirm the correct functioning of the Perl interpreter. Arguments are passed to the script using the address bar of a web browser, and in a section entitled "Web Browser as Attack Tool", NMRC explains how to suborn the script. Instead of displaying path names and files relevant to CGI script functionality, in the example given, the script is directed to list the contents of the /etc/passwd file used on a UNIX server for storing passwords (Nomad Mobile Research Centre, n.d.).

NMRC Example #2

Another example from the same source details how to exploit another script that is commonly installed by default:

> The phf file is an example CGI script that is used to update a phonebook style listing of people. By default, a lot of sites have this file sitting in /cgi-bin and don't even know it. You know, they installed everything to default. However, the phf file behaves "differently" if thrown a newline (0a) character. Here's the common attack for a Unix server:
>
> http://example.com/cgi-bin/phf?Qalias=x%0a/bin/cat%20/etc/passwd
>
> We've used a Unix target as an example since it is most common, but NT commands will work on a NT server just fine, too. (Nomad Mobile Research Centre, n.d.).

B0iler and Rain Forest Puppy

Another site provides a lengthy article entitled "Hacking CGI - security and exploitation", which goes into more theoretical detail but also provides examples of specific exploits. The author, identified only as b0iler, gives credit to another hacker known as Rain Forest Puppy as the originator of much of the information in the article. However, b0iler felt there would be a benefit to recapping that information because "although that paper was brilliant… it is a very hard tutorial for a newbie to understand" (b0iler, 2002).

Nikto

Rain Forest Puppy is perhaps best-known for having authored whisker, a vulnerability scanner for Perl scripts that is, itself, written in Perl. Whisker is no longer maintained, but has been replaced by a similar program called Nikto, which gives credit to whisker as a predecessor and, in particular, the basis of its libraries. In the words of is developers, Nikto "performs comprehensive tests against web servers for multiple items, including over 3200 potentially dangerous files/CGIs" (cirt.net, n.d.). Nikto can tell you if your web server (or someone else's) contains scripts with known vulnerabilities, such as the ones detailed above. Like many security tools, it is double-edged, of use to both malicious hackers and security-conscious network administrators.

SOLUTION #1: OUTSOURCING THE HEADACHES

For the website designer who wants to include dynamic code but does not require complete control over the details, the best solution may be to have any interactive content hosted on someone else's server. It is almost certainly the best solution for many small and medium-sized businesses, who want to sell merchandise online while minimizing the risk of an embarrassing compromise of customer privacy. For individual web authors with minimal skills, designing a personal page to be hosted on web space provided by their ISP, it may be their only alternative if they want that guest book or hit counter. Many such web space accounts, not only those provided by ISPs but also other lower-priced web hosting services, have no provisions for running CGI scripts.

eFreeGuestbooks

I was discussing the topic of this paper with a friend, who maintains several very basic personal pages. He was alarmed at what I was telling him, because he had recently added a guestbook to his site and wanted to know if this could allow a hacker to alter his text or images. After a quick look at his source code, I was able to reassure him that the guest book did not appear to provide any potential access to his site. The code, which he got from the guest book provider, redirects the browser to their own site:

> Sign my Guestbook (eFreeGuestbooks, n.d.).

Note that the URL path, before the question mark, ends with the .pl extension; almost certainly a Perl script. But any vulnerabilities are not the concern of the web designer. Any successful hack would give the attacker access to the efreeguestbooks.com server, not the designer's pages.

Drawbacks of Outsourcing

This service is designed for those with little knowledge of, or interest in, programming. Like most outsourced scripting, there is a price for the guestbook; the user must either pay a nominal fee of $12.95 per year, or else accept a banner ad at the top of the guestbook page. What will prove more annoying to anyone operating beyond the most basic level of page design is the lack of control. The user has some discretion over cosmetic issues such as the background and text color (eFreeGuestbooks, n.d.). But beyond that, the web designer will have no control over the appearance of the guestbook. It is the other cost of making script security into someone else's problem; you are no longer in the driver's seat. You are just along for the ride.

SOLUTION #2: USE CODE WRITTEN BY EXPERTS

For the web designer who wants more control over the appearance and behavior of a dynamic page, numerous scripts are available for free download. Since Perl is an interpreted language, the code is right there to be modified and analyzed to the limit of the user's ability. This presupposes, of course, that the server hosting the site has the capability to run CGI scripts. That is not hard to accomplish, however. Many hosting services offer that functionality, and for those hosting sites on their own machines, the Perl interpreter is not difficult to install. As a cautionary note, it is worth emphasizing that in the latter case, the data that could be compromised in the case of a vulnerable script is no longer limited to web content but now includes all information on the designer's network.

Matt's Script Archive

One of the most popular collections of free Perl scripts is Matt's Script Archive, found at www.scriptarchive.com. Mr. Wright offers over a dozen of his own scripts, as well as offering a showcase for a few scripts written by others people. Mr. Wright relates on his site that he began writing Perl scripts over 10 years ago, while still in high school. He explains, "Matt's Script Archive was one of the first sites on the Internet to offer Free CGI scripts that were well-documented and easy to install.... Having no formal programming education or experience at that time, I released the code as I learned, hoping others would find it useful" (Wright, n.d.). The scripts he provides range from simple hit counters and guest books to a search engine, a discussion board and FormMail, probably the most well-known and popular of Matt's scripts. The scripts are no longer actively maintained, for the most part, but remain very popular (Wright, n.d.).

NMS

In 2001, a group of programmers known as the London Perl Mongers became alarmed and annoyed at what they considered the shoddy and dangerous programming practices used in Matt Wright's widely distributed scripts. They founded the NMS project, and while they are a bit coy about what that stands for, one can easily guess. The group explains their purpose as follows:

> The problem is that the scripts in Matt's Script Archive aren't very good. The scripts are well known amongst the Perl community to be badly written, buggy, and insecure. Anyone asking for support on Matt's scripts in any forum will be told in no uncertain terms that they shouldn't use his scripts.
>
> Unfortunately for some time there were no replacements for Matt's scripts that you would want people to use. In 2001, the London Perl Mongers decided to address this problem and write a series of drop-in replacements for Matt's scripts. This project is the result. (NMS, n.d.).

One might assume that Mr. Wright would be either embarrassed or defensive—and that he might well ignore this less-well-known alternative to his work. On the contrary, he devotes a page of his site to their work, and recommends their programs. He acknowledges that his own scripts have their flaws, suggests that the NMS scripts are better models for beginning programmers, and adds " The code you find at Matt's Script Archive is not representative of how even I would code these days." (Wright, n.d.). Rather than providing a link to their site, for some reason, he has created a mirror with copies of their scripts and appears to be diligent about updating it. For their part, the programmers of the NMS project have added a tip of the hat to Mr. Wright on their own site. They do not disown their critique of his programming, but take care to state that they have no wish to be personally insulting, and to that end, are rather vague about what the acronym NMS actually stands for (NMS, n.d.). I assume it originally meant 'Not Matt's Scripts', but at this late date and with a certain amount of mutual respect having been established, one of the discussion boards on the NMS site has a thread for alternate interpretations of the name, with some amusing and creative suggestions such as one might expect from the kind of bright and quirky people who enjoy programming so much that they do it for no monetary compensation.

SOLUTION #3: BEST PROGRAMMING PRACTICES

This paper is not intended to be a Perl tutorial, nor am I qualified to write one (though for those seeking such instruction, it is readily available online). It is, however, necessary to have at least some knowledge of Perl programming in order to evaluate the security of unfamiliar scripts and to modify any script—and when doing such a modification, it is also important to keep security in mind lest the tailored script should introduce vulnerabilities not found in the original. Examining well-written code is an excellent way to learn, and as mentioned above, the NMS scripts are widely regarded as an excellent model.

Evaluating Perl Scripts

As Bruce Schneier points out in his excellent book, "Secrets and Lies", security is much harder to evaluate than functionality. It's relatively easy to put a program through various paces and see if it crashes or returns an unexpected result. Evaluating the security of a program is not only harder, it's impossible to ever be sure it is completely secure—a question of proving a negative. The best one can do is to demonstrate that the software can only be hacked by someone who thinks of something the tester did not (Schneier, 2000). Dismaying as that may be, there are several best practices that the Perl programming community has learned through experience will help make scripts more secure.

Dave Cross provides a security checklist for Perl scripts on www.perl.com. which is owned by the well-respected programming publisher, O'Reilly. Although the

list is intended primarily for those evaluating an existing script, it is a good resource for the beginning programmer as well. He recommends checking to see if the script uses the –T and –w switches, as well as the 'use strict' command and the CGI.pm module (Cross, 2002).

It is relatively easy to see if most of these suggestions have been followed, since three of the items mentioned above would be near the top of a script that includes them. The following line, known as the "shebang" line of a Perl script, will be the first line of any Perl program and includes the path to the Perl interpreter:

```
#!/usr/bin/perl -wT
```

Note that two of the switches used above are included, combining the warning feature and the Taint mode. The –w switch will instruct the interpreter to "issue warnings for questionable practices" (Wall, 200, p. 137). Taint mode will "tag" data as having originated from user input, and will treat that data differently than if it were obtained from a more trusted source such as a database on the web server.

The "use strict" command enforces good programming practices, and CGI.pm is one of many plug-in modules available for Perl. As Mr. Cross explains,

> This module contains a number of functions for handling various parts of the CGI protocol. The most important one is probably param, which deals with the parsing of the query string to extract the CGI parameters. Many CGI scripts write their own CGI parameter parsing routine that is missing features or has bugs. The one in CGI.pm has been well-tested over many years in thousands of scripts - why attempt to reinvent it? (Cross, 2002).

Lists of Links: A Comparison

In light of this information, I thought it would be interesting to compare the security features of some scripts that have a similar features. I chose the "Free For All Links" script from Matt Wright, and the identically-named alternative from NMS. Both are short, simple scripts that allow users to build a list of web links. This set of scripts caught my attention because I wrote such a utility myself, although the best thing that can be said about my effort is that it makes Mr. Wright's product look very professional by comparison.

The Matt's Script Archive "Free For All Links" script is a little over three pages long when printed out, including a large commented-out area with credits and a copyright notice dated 1996. The NMS version has only sparse and functional comments, but weighs in at just over seven pages printed out in the same font. Much of the additional code is error handling and data checking, which in Perl

often ends with the ominous sounding imperative "or die"—directing the script to stop running if some particular condition is not met.

As advised in the section above, the shebang line of the NMS script uses both the warning and Taint mode switches, and includes "use strict". (NMS, n.d.). The Matt's Script Archive version of the same script does not use any of these (Wright, n.d.). For user input, the NMS "Free For All Links" uses the following code:

my $url = param('url') || ''; (NMS, n.d.).

Use of the "param" function indicates that the CGI.pm module is in fact being invoked, and an empty value is supplied to initialize the variable $url in the absence of user input. Note also the use of 'my' before the variable name, which limits the scope of $url to a local variable. In his version of the script, Mr. Wright's script uses the STDIN method of accepting user input, which is more familiar to me but is apparently not the choice of more highly skilled programmers.

CONCLUSION

I had hoped that the research involved in writing this paper would leave me feeling more confident in my decidedly rudimentary Perl programming skills. In fact, it has impressed on me quite firmly how little I actually know. Reading over Matt Wright's script, I understood most of it and was able to fill in the gaps by researching commands and syntax that I had forgotten or never learned. Mr. Wright's skills as a Perl programmer are definitely a step ahead of my own, even using examples from over ten years ago when he was in high school, but I felt that I could get to that level with a modest amount of work. Unfortunately, even Mr. Wright himself agrees that is a very poor example to follow. I learned from studying the NMS script, but still understand considerably less than half of what is going on. I do not feel at this point that I am ready to start writing scripts that have any chance of public exposure, and the amount of work I would have to do in order to reach that level of expertise seems daunting.

Installing an existing script is not difficult, and I hope this paper has offered some guidance on how to select one that is well-written. The NMS scripts are well-commented, particularly in the section that is meant to be customized by the end user, and that section is clearly marked with the comment " # No user maintainable parts beneath here" (NMS, n.d.). But for some of us, the urge to play with the code that lies beneath that notation may prove to strong to resist. And modifying well-written code is an excellent way to learn. But I would strongly recommend that before taking that modified script live on the internet, the programmer should exercise great care to be sure he or she understands the purpose of the original code and the implications of those changes.

REFERENCES:

b0iler. (2002). *Hacking CGI - security and exploitation*. Retrieved April 30, 2006 from http://www.hnc3k.com/hackingcgi.htm

cirt.net. (n.d.). *Nikto 1.35*. Retrieved April 23, 2006 from http://www.cirt.net/code/nikto.shtml

Cross, D. (2002, January 23). *Finding CGI Scripts*. Retrieved April 19, 2006 from http://www.perl.com/pub/a/2002/01/23/cgi.html

EFreeGuestbooks. (n.d.). *Fully customizable guestbooks for each user*. Retrieved June 7, 2006 from http://www.efreeguestbooks.com/features.html

nmrc.org. (n.d.). *Web Browser As Attack Tool*. Retrieved April 23, 2006 from http://www.nmrc.org/pub/faq/hackfaq/hackfaq-09.html

NMS. (n.d.). *Web programs written by experts*. Retrieved June 8, 2006 from http://nms-cgi.sourceforge.net/about.html

Schneier, B. (2000). *Secrets and Lies: Digital Security in a Networked World*. Indianapolis, IN: Wiley Publishing, Inc.

Wall, L., Christiansen, T., & Orwant, J. (2000). *Programming Perl, Third Edition*. Sebastopol, CA: O'Reilly & Associates, Inc.

Wright, Matt. (n.d.). *Matt's script archive*. Retrieved June 8, 2006 from http://www.scriptarchive.com