Global Information Assurance Certification Paper

Secure Programming
"The Foundation to Secure Computing"
Jeff Bowes

Introduction

With the boom of the Internet, security has become a big concern for most of us that connect our systems to it. We try diligently to tweak our systems to secure as many of the vulnerabilities as we can. For those of us writing software, we can make this job easier by taking the time and responsibility to make our software more secure. However, many of us are pressured to make deadlines that don't allow us to take this time. Needless to say, we'll be doing it anyhow later as we fix the holes that some hacker has exploited. Whatever the reason is for not taking the time to make your code secure, the result will be a very unhappy boss or customer.

This paper presents some ideas and insights into ways that make programs more secure and shows how to avoid some of the more common pitfalls. Hopefully, you will get a better understanding of what it takes to make your code more secure.

Why the need to program securely?

To get a better understanding of why our programs need to be secure, we need to understand the three primary computer security goals of confidentiality, integrity, and availability.

- Confidentiality means that the computer systems' resources, files, etc are only accessible by authorized users.
- Integrity means that the computer systems' resources, files, etc are only modifiable by authorized users in appropriate and authorized ways.
- Availability means that the computer systems' resources, files, etc are available to authorized users in a timely manner.

With these in mind you will better understand the reasons for secure programming and also ways to better protect these in your code.

What kind of programs need to be done securely?

All programs should be developed as secure as possible. Most of the techniques to secure programming are simple good programming techniques to begin with. Good secure programming practices should be applied to the following types of programs:

- setuid and setgid programs.
- network daemons.
- programs that require atomicity.
- programs that get input from outside or use information from the environment.
- system administration programs.

How are non-secure programs a threat?

There are three ways programs that weren't written securely can become a threat. The first type of threat is buffer overflows. When this threat is exploited, the attacker is capable of breaking all of the basic security goals. Buffer overflow attacks are aimed at programs that take data as input and place the data in buffers, but with out checking to see if the buffer is large enough for the data. The attacker sends to much data with there own embedded commands hoping to have the commands executed when the system overwrite the area in which the code was. Originally depending on the program, the attacker could now have privileges that exceed their normal privileges including those that give them complete control of the system. Some programs just crash denying further access to the service or bring the entire system down, denying access to any services provided by the system.

The second type of threat is race conditions. This is a behavior that involves a shared resource such as a file or variable where multiple access has not been properly programmed. These can be caused by trusted or untrusted programs. This type of threat has the ability to give an attacker access to files they wouldn't normally have access to.

The third threat is setuid or setgid programs. These programs run with the privileges of a specific group or user not necessarily the user that is running them. These types of programs can suffer from buffer overflows and race conditions as well giving a potenial attacker privileges they might not have had. With these things in mind we'll look at ways to program more securely and tips you can use to ensure this.

What ways can I program more securely?

General tips

Be paranoid, assume anything you code will bring about inquiring minds who will try to break it, and make it do things it wasn't meant to do. Even if they don't, you can always assume that the intended user will supply erroneous inputs that will cause it to do things unexpectedly.

- Make sure you have a clear understanding of what you are trying to code.
- Avoid complex code, keep it short and sweet with just what is necessary to get the job done.
- Check every input provided by the user.
- Check every return code from functions as well as those from other libraries.
- Don't depend on data hiding to make programs secure.
- Use the least sufficient privilege necessary to do the task and only when the privilege is needed.
- Use as few shared resources as are necessary.
- Reuse previously tested code if it fits with what the program is doing.
- Don't trust common libraries to necessarily be secure.
- Check all arguments passed to system calls.

- Always use full pathnames.
- Always set the current working directory yourself.
- Use some form of load limiting to prevent overloading of your server.
- Don't create files in world-writable directories.
- Don't have your users send reusable passwords in plaintext.
- Make good use of tools such as lint.
- Use reasonable timeout limits on network reads and writes.
- Thoroughly test your program.
- Have another competent programmer review your code.
- Statically link your programs.
- Trap every signal.

These tips should work for most programming languages. The tips will go a long ways in helping to write more secure programs. Below are language specific tips that will further help to write more secure programs.

Tips for C

C is one of the most widely used languages for network programming. Unfortunately it also has a lot of standard routines which are actually very insecure. C is quite powerful and sometimes you can make code do things you never thought it could do. C is also much harder to write secure programs when compared to Perl. It doesn't have the automatic memory management that Perl has. Here are some tips for writing programs in C.

- Use subroutines that do bounds checking, fgets(), strncpy(), strncat() versus using ones that don't like gets(), strcpy(), strcat().
- Never use the subroutine system(), use exec() instead, but avoid execlp() and execvp().
- Avoid using the popen() system call.
- Generate as many warnings as possible. Use -Wall option if you are using the GNU C compiler.
- If you are creating new files with open(), use the O_EXCL | O_CREAT flags to cause the routine to fail if the file exists.
- When creating temporary files use tmpfile() or mkstemp() routines.
- Use lstat() to check that a file is not linked.

Though C is in reality a very unsecure language to write programs in, with some practice and by following these tips you can go a long ways to make programs secure.

Perl Tips

Perl is a little easier than C to write more secure code. It has the advantage of built in memory management. But it still has its share of pitfalls. Here are some tips that will help to better your Perl programming skills.

- Use the Perl tainting features by using the -T switch.
- Perl ignores the tainting for filenames that are opened read-only. But you should still ensure that you un-taint all your filenames.
- Use Perl's emulation mode for handling setuid scripts.
- Always set your program's PATH environment variable.
- Be sure the Perl interpreter and its libraries are only modifiable by the administrator.
- Be careful of using the *eval* statement.

UNIX Shell Tips

Many times using a powerful language like C or an interpreter like Perl are unnecessary and can be handle using simple shell scripts. But sometimes these simple shell scripts can be the bane of our existence when someone uses them improperly. Here are some tips that will help minimize these problems.

- Do not use the Bourne shell for setuid/setgid scripts.
- Set PATH to a known value.
- Set IFS to a known value.
- Do not use shells (sh, tcsh, csh, ksh or bash) for CGI scripts, unless they are very trivial.

Tips for Setuid Programs

Many programs need to run with user permissions different than that of the current user running it. This is done by making the script setuid or setgid. These programs unfortunately can be come real security problems. Below are a list of ways to help keep these types of programs from becoming a problem.

- Avoid using superuser unless superuser is the only user that can perform these functions.
- If the program requires a portion of it to run as superuser, put the SUID part in a different program and construct a carefully controlled and monitored interface between the two.
- Use these privileges that are needed when they are needed and return back to the original privileges as soon as they are no longer needed again.
- Avoid writing SUID shell scripts.
- Use full pathnames for all files that you open.
- Use chroot() to restrict the program to a specified subdirectory within your filesystem.
- Bracket sections of code that require higher privilege with setuid or setgid functions.

How do I ensure the programs are secure?

Always test the program to ensure it is secure. Try to think like a hacker, imagine what a hacker might do to try and break your code. Here are some ways to test your program and help reduce security holes in it and improve the quality of your code.

- Try to overflow every buffer.
- Try to abuse all the command-line options.
- Try to create race conditions.
- Have someone review and test your code.
- Read through the code, try to find parts that might be vulnerable.

After trying these you'll get a great appreciation of the code you have written and the efforts you have put forth to ensure there aren't any glaring security holes in the programs you produce.

If you take the time to make your code secure, it will help ensure that the main goals of computer security are maintained. It will also make it easier for those of us less savvy in the realm of computer security more able to protect ourselves from the threats on the Internet. Writing secure code is not just about making it hack proof, its about applying simple proven programming techniques to produce code that is sound.

References:

Al-Herbish, Thamer. "Secure UNIX Programming FAQ." Version 0.5. 16 May 1999. URL: http://www.whitefang.com/sup/.

Wheeler, David. "Secure Prgramming for Linux and Unix HOWTO." Version 2.75. 12 Jan 2001. URL: http://www.dwheeler.com/secure-programs/ .

Galvin, Peter. "The Unix Secure Programming FAQ." "UNIX Insider". URL: http://www.sunworld.com/sunworldonline/swol-08-1998/swol-08-security.html.

"Writing secure CGI scripts". "The Common Gateway Interface". URL: http://hoohoo.ncsa.uiuc.edu/cgi/security.html .

"NCSA Secure Programming Guidelines". URL: http://www.ncsa.uiuc.edu/General/Grid/ACES/security/programming/ .

Bishop, Matt. "Writing Safe Setuid Programs". 30 Nov 1999. URL: http://seclab.cs.ucdavis.edu/~bishop/secprog.html .

Bishop, Matt. "Robust Programming". URL: http://seclab.cs.ucdavis.edu/~bishop/classes/ecs153-1998-winter/robust.html .

Garfinkel, Simon and Spafford, Eugene. "Secure CGI/API Programming". "WebServer OnLine Magazine." July 1997. URL: http://webserver.cpg.com/features/cover/2.6/ .