Global Information Assurance Certification Paper

Rishona Phillips
GIAC Security Essentials Certification (GSEC) Practical Assignment
Version 1.4b – Option 1

The Bugs are Biting

Abstract:

Currently there are thousands and thousands of software programs running on millions of computers across the world. Yet there are also bugs crawling around the code of these programs that are biting holes into the security of the machines they run on. Unless this issue is seriously addressed in the near future, one can expect to see an increase in the severity and quantity of security breaches. This paper will give a general overview of the problems and challenges of software mistakes and how they affect security.

Introduction

How often would you buy a $300 product and keep it after you found out it was defective? Many of you may be thinking well that's kind of a silly question, of course I would bring it back! While this may be a cut and dry situation for the majority of consumers, this does not hold true in the computer software industry. Software Bugs have become synonymous with many of the major commercial and open source applications available today. It is not uncommon for the public to actually expect to have bugs in operating systems, databases, games, mail programs, and the list goes on.

History

The first computer bug, a moth, was found in a Mark II computer back in 1945. Even though this bug (of the insect variety) was not an actual software mistake, it seems that the term stuck and is now widely used today to describe things that do not work in software. A Bug is an "unwanted and unintended property of a program or piece of hardware, especially one that causes it to malfunction"(techditionary).

This term is used all too often, and it seems that many people have overlooked what the term computer bug really means. It is often forgotten that a "bug" is the result of a programming error and not another feature of their program.

According to CERT there were six security incidents/vulnerabilities reported in 1988; ten years later in 1998 there were 3,784, and in 2002 there were 82,094. Unfortunately the amount of reported "bugs" is growing by leaps and bounds, while the amount of confidential and supposedly "secret" information is also being kept in a web accessible computer.

Common Coding Mistakes

According to David Wong's article Secure Code; buffer overflows, format string vulnerabilities, authentication, authorization and cryptography make up 90% of the current security vulnerabilities. While this may not be new information, the fact that the percentage is this high should be a cause for concern, considering these errors are usually human in nature.

Following are a few of the main programming errors that cause most of the security vulnerabilities in software programs. Once exploited these coding mistakes can cause serious damage to a compromised system. Unfortunately these are also errors that could have been mitigated earlier on in the development process.

Buffer Overflows

A well-known and frequently exploited problem in code is the “buffer overflow” or “buffer overrun” which allows an attacker to gain root or administrator access to a machine. A buffer overflow is simply an overflow of information that runs out of the buffer and into RAM, and is then dumped into the program code and executed. As a simple example, let’s say that a certain section of a webpage requires a user to enter in ten characters of information, but instead the person enters fourteen characters. If this is an exploitable buffer overflow, these extra bytes end up being dumped into stack memory. This becomes a problem when malicious code overwrites the return address in memory and then runs from the memory of a program, which gives an attacker full control of a system or network.

Due to the fact that buffer overflows run rampantly through many software programs finding an exploit of this is not a difficult task. One of the most well known exploits of a buffer overflow was the Morris Worm that spread across the Internet on November 2, 1988. Although Robert Morris Jr. did not have malicious intent, his unintended clogging of the Net did wreak serious damages.

Yet Morris’ worm also served as a wake up call to security professionals everywhere. The worm caused “perhaps 10 percent of the computers on the Internet to be infected” all because of a coding mistake (Sullivan). Surprisingly Morris did know about his buffer overflow mistake and released the worm to the Internet to exploit it. In any case the worm definitely caught the attention of those using the Net especially those in the security industry.

.

Format String vulnerabilities

A format string vulnerability is a programming error that has “lingered around software code for years” although “the risk had not been evident until mid-2000” (McClure, Scambray, and Kurtz 324). Basically format strings occur because a programmer has chosen to use incorrect functions that do not properly handle strings.

Window’s New Techonlogy 4.0 and Windows 2000 had a very serious problem with format string vulnerabilities and the drivers. The functions within the Windows device drivers handled formatting string characters incorrectly. This vulnerability was harmful because “this bug can lead to the possible patch of the kernel memory”. In other words, someone can install a backdoor on this system and now have control over any files, folders or installs on this machine.

Heap Overflows

Heap overflows are also in the same class as buffer overflows except for one small difference. The buffer overflow attack overwrites the stack memory where any program can store its information, whereas heap overflows overwrite the memory addresses that are specifically allocated to that program. In the end an

attacker can do the same thing as in a buffer overflow, just in a slightly different manner.

Just recently (January 25,2003) the Slammer/Sapphire worm hit the Internet with amazing speed because it “doubled in size every 8.5 seconds” and “infected more than 90 percent of vulnerable hosts within 10 minutes” (Moore, David et al.). Slammer affected Window’s 2000 SQL Server Resolution Service SRS, which in a normal environment “listens on port 1434 and returns the IP address and port number of the SQL server instance that provides access to the requested database” (Vulnerability Note VU#399260).

Yet because there was a heap overflow vulnerability in SRS an attacker had the ability to send a bogus request to port 1434 and have their code run inside the SQL server as a legitimate account. Even though there was not a malicious payload in the Slammer/Sapphire worm, the damages it caused for unpatched SQL Servers and the clogging of the Internet took days to completely clear up.

Even though many System Administrators did not take the time to patch a known vulnerability, this is another example of a software mistake that caused serious damages.

Issues with Programming Languages

It is no secret that the programming languages of today are getting more powerful and more capable of creating complex programs. Yet as these languages advance, the need for security also increases. Unfortunately many of the most popular coding languages are often where poor security measures are first implemented. As security is not always a major priority at this stage the potential for mistakes and poorly coded programs is coupled with programming languages that were not all written with security concerns in mind.

Most programming languages were first created long before computers and the Internet became as popular as they are today. Their creators did not anticipate the amount of e-commerce, email, online banking, and e-business utilized by millions of people and business’ around the world.

C

With the invention of the C programming language came many new possibilities and many overlooked security concerns. Dennis Ritchie created the C programming language at Bell Labs in 1972. C was created as a successor to the B programming language. Ritchie and his team (Thompson and Kernigham) then used the newly created C compiler to rewrite the UNIX operating system.

The C programming language is a very old but still powerful and widely used tool. It is used to build a variety of applications ranging from games to operating systems. It is still the basis for some of the most popular programming languages of today such as C++ and Java. Although C is still very popular it contains many pitfalls. The problem with C is that it has no safeguards. As long as the code is syntactically correct, the code will still be compiled regardless of its purpose.

This is how certain applications have become vulnerable to buffer overflows. For instance, by declaring an array in a C program there is a possibility to create a loop that tries to write beyond the array's declaration. The program will then compile without any warnings or errors because there is nothing wrong with the syntax. Once the program is run the code with the loop will start overwriting memory that it should not. This has been the cause of many buggy programs ranging from minor nuisances to major security breaches. Many malicious programmers take advantage of C's power and purposely write malicious code to exploit a program.

C++

C++ is one of the most popular object oriented languages in use today. It was created by a man named Bjarne Stroustrup in 1985 at Bell Labs to be the successor to C. C++ still maintains many details of the C language and will allow everything that C does and more. Yet, C++ is basically object oriented C, therefore it inherited many of C's problems.

Java

Java is an object-oriented language that became popular because of Sun Microsystems push to prove that their technology was the best. Java was supposed to provide a very stable object oriented programming language with security as a main priority. Java programs were supposed to run inside a protected environment, which would keep hackers from exploiting badly written software. Although this sounds nice Java still encountered many problems, as there were holes in the Java programming language itself. This became apparent during the very popular era of programs called hostile Java apps.

The Java security model contains the details of the security features within the language itself. Before a Java program is loaded and run, it takes its trip through the verifier. The verifier checks for common security problems within the program. It checks to see if the file is in the correct format, whether the file contains any stack overflows and if there are any illegal type casts. A type cast is the conversion of one type to another, for instance the conversion of a decimal number to a string. The verifier does a four-pass system check. It checks for any of these security flaws in four passes on the file. After the file is verified, it is then sent through the classloader. The classloader's job is to load the class after it has been verified.

Once loaded, the next job is for the security manger to prevent a Java program from accessing system resources. If a Java program tries to access a local file or network resource, it goes through the security manager. The security manager either allows or denies the operation. The actual decision-maker is the access controller. It is a very complicated piece and in conjunction with it there exists a policy object that stores user's settings for permissions granted to classes. All of these act as the sandbox model that Java is supposed to represent. The sandbox is a container within the system where the Java program is contained and the program cannot access anything outside of the sandbox. Although Java has received some skepticism, this method seems to be working.

Perl

Perl (Practical Extraction and Report Language) began in 1987 as a cool tool that produced reports in a hierarchy for a bug-reporting system. Even though it had a very simple beginning, it became a powerful and efficient language that many programmers started using for other programs that it wasn't initially designed for. As the years passed Perl did things that nobody had ever envisioned.

Although Perl is one of the most popular programming languages especially for writing CGI scripts, it still has some security implications. To deal with all security threats that are a threat by the Internet, Perl uses taint mode. Taint mode will treat all data as bad unless otherwise told that it is not. If bad data was entered onto a form and the CGI script tried to pass the form data to a system call the taint mode would not allow it. Taint mode protects from this and other security issues.

Visual Basic

Visual Basic was the successor of the BASIC programming language. Microsoft has marketed this technology many times. Visual Basic is a very easy to use and a simple language to build computer programs. A lot of intricate details are hidden from the programmer but this does not change the fact that there are still possibilities for security implications with using Visual Basic. Visual Basic and any other Windows applications are as vulnerable as a program written in C. The Cult of the Dead Cow brought this to light with their article called "Tao of the Windows Buffer Overflow."

In the world of programming languages programmers should respect the public documents that go into explicit detail on what not to do in a programming language. Too often these warnings go unheeded, hence many of the security issues found in programs today.

Open Source vs. Closed Source

The endless debate of whether closed or open source software is more secure continues to be a source of disagreement depending on which camp one resides in. Both sides have legitimate arguments for why their open/closed program or application is better, although the open source world does have a more compelling argument. The often used quote “security through obscurity” receives its fair share of criticism from open source advocates because they feel that making code a secret limits the security of the application or program. The closed source programmer’s argue that allowing the public to see the code, will also allow malicious attackers an easier way to exploit the system.

Michael Warfield’s article has offered some interesting insight into why open source causes programmer’s to code more securely. One of the most convincing aspects of his position is that the open source programmer knows that what he is releasing to the community will be reviewed by his peers. Warfield says “ the open source programmer, puts his reputation on the line with every line of code he writes” (Warfield). This knowledge that other people in the industry will be aware of mistakes, ups the ante a bit for people contributing to open source projects.

Warfield also talks about the fact that closed source programmer’s are often out of the loop when it comes to secure programming and best practices. Unfortunately too many closed source programmers work on projects that are a “company secret” and cannot be discussed with others.

A very successful and secure Open Source project was started by a man named Theo de Raadt is OpenBSD, which he began after leaving the development of the NetBSD project. The team he assembled took the UNIX operating system and did a line-by-line audit of the operating systems source code. This audit was started in 1996 and de Raadt and his team found many bugs. After finding more bugs in the source code they finally decided to modify the code.

Another successful open source software project is the Samba project. Samba implements the SMB protocol on a Linux or Unix machine. Samba acquired it’s name by taking the letters of the acronym SMB (Server Message Block) and placing “A’s” within the sequence to allow for better pronunciation. Although Samba has had some security flaws they have been fixed quickly and efficiently due to its status as an open source project.

Who’s to Blame?

The number of flaws in software code is a disappointing yet undeniable fact today. Unfortunately too few people get angry with the companies who develop these faulty and “mistake” ridden programs to the public. There are many instances of poorly written code being exploited by a malicious hacker, yet one has to wonder if the software companies should also be held accountable for a product rife with errors.

Even though major security vulnerabilities in popular software programs receive a lot of media attention, the average person does not know that most of the problems with software are the company’s fault. Many non computer savvy users first hear about a new exploit because of the attention the media gives to the crackers(s) or hacker(s) who compromised the system or network. While there is no argument that a hacker or cracker should be prosecuted for breaching system or network security one may wonder why there are no consequences for the software companies.

If there were stricter guidelines and rules for the product released to the public, the potential for security vulnerabilities would drop. Yet, while it is virtually impossible to release software that is mistake free it is not impossible to get better at debugging and testing during the development process. These are some of the most important steps in programming, and are oftentimes the most poorly initiated.

Aside from the all too familiar complaints that follow the release of a major software vulnerability; software companies do not receive any other repercussions. Considering that most of the popular corporations monopolize the market today there will not be any changes in the software until the companies are required to adhere to standards in the software they release.

Microsoft

Although Microsoft is the most widely used operating system worldwide, it also receives the most criticism regarding its software vulnerabilities. Considering that a Window’s OS has over a million lines of code, debugging and error detection is not a picnic. Yet and still, the amount of critical software mistakes in their code that have allowed malicious code to exploit their vulnerabilities can not be ignored.

Microsoft’s president and CEO Bill Gates has made a new commitment to securing his OS, which he calls a “war on hostile code” (Lemos). Over the next nine years (the first one has already passed) Microsoft’s goal is to win back its customers trust and respect with the “Trusted Computing” campaign. At this point in time it is too early to tell if they will successfully accomplish this feat.

Unfortunately they are still being hit with software mistakes, even though they “halted product development” to train their developers on coding more securely

last year (One year on is Microsoft 'trustworthy'). As a result they threw out most of the "Windows code" and lost about $100 million.

Oracle

In light of CERT's release of 37 "separate security holes of varying seriousness" last spring, Larry Ellision might not want to declare an "unbreakable" database any longer (Gardener). With most of the 37 "bugs" stemming from code mistakes, Oracle is looking very breakable from a security standpoint.

Judging from Oracle's clientele ranging from General Dynamics, to Visa and the Ministry of Defense, it would be safe to guess that these customers rely on Oracle to provide them with superior security. Judging from the type of work these organizations/companies produce, it is a simple deduction that a security breach could be a potentially harmful situation.

Linux

Even though there are many positive benefits in open source software, it is not immune to bugs. According to Fred Langa Linux's growing popularity as a mainstream product will cause "more problems" to "come to light" and the bug reports to increase (Langa, 1).

His reasoning behind this is fairly simple and is due to the fact that Linux started out as an operating system for advanced computer users. Currently Langa says that mostly anyone can get their hands on a Linux machine, but unfortunately most ordinary people really don't know enough about Linux to run it securely. This along with the rising bug reports coming from popular flavors like Red Hat are making Linux a playing field for security breaches.

Conclusion

Software mistakes will never disappear from software, but they can be reduced at an earlier time in the process so there are fewer critical security problems. There are many programming tutorials and books that specifically deal with coding best practices, yet too often best practices are overlooked for time constraints or "the easy way out". On the other hand there are many good development teams and programmers who do care about the quality of their code and that may one day be the deciding factor for which companies make money off software programs. Eventually consumers will realize that they are being short-changed and demand a product that does not need five critical security patches a week.

Works Cited

CERT. “Vulnerability Note VU#399260”. URL: http://www.kb.cert.org/vuls/id/399260 (22 Apr. 2003).

“CERT/CC Statistics 1988-2003”. URL: http://www.cert.org/stats/cert_stats.html (20 Apr. 2003).

DilDog. “The Tao of Windows Buffer Overflow”. 1 Mar. 1998. URL: http://www.cultdeadcow.com/cDc_files/cDc-351/ (19 Jun. 2003).

Gardener, Joey. “Oracle named and shamed for security problems” 15 Mar. 2002. URL: http://www.silicon.com/news/500013/1/1032058.html (20 Apr. 2003)

Kolishak, Andrey. “NT Drivers Potentially Vulnerable to Format String Bug (FSA Bug)”. 24 Apr. 2002. URL:http://www.securiteam.com/windowsntfocus/5LP0N1F41O.html (22 Apr. 2003).

Kerer, Clemens. “How the Java Security Model Works”. 22 Sep. 1999. URL: http://www.infosys.tuwien.ac.at/Teaching/Finished/MastersTheses/JSEF/node27.html (14 Apr. 2003).

Langa, Frank. “Langa Letter: Linux Has Bugs: Get Over It”. 27 Jan. 2003. URL: http://www.informationweek.com/story/showArticle.jhtml?articleID=6512225&pgno=1 (19 Jun. 2003).

LeClaire, Jennifer.”Microsoft and the New Science of Security Flaws”. 26 Sep. 2002. URL: http://www.ecommerctimes.com/perl/story/19514.html (17 Apr. 2003).

Lemos, Robert. “Microsoft declares a “war on hostile code”. 10 Apr. 2001. URL: http://news.com.com/2100-1001-255638.html?legacy=cnet (20 Apr. 2003).

---. “One year on, is Microsoft ‘trustworthy’?”. 16 Jan. 2003. URL: http://news.com.com/2100-001-981015.html?tag=nl (20 Apr. 2003).

McClure, Stuart and Joel Scambray, George Kurtz. Hacking Exposed: Network Security Secrets and Solutions, Third Edition. Berkely, California, Osborne, McGraw-Hill, 2001. 324.

Moore, David et al. “The Spread of the Sapphire/Slammer Worm”. URL: http://www.cs.berkeley.edu/~nweaver/sapphire/ (22 Apr. 2003).

Olavsrud, Thor. “Multiple Security Flaws Found in Oracle Servers”. 15 Mar. 2002. URL: http://www.internetnews.com/dev-news/article.php/10_992381 (17 Apr. 2003).
Perl mongers. “Perl History”. 12 Aug. 1999. URL: http://www.perl.org/press/history.html (19 Jun. 2003).

Schwartz, Randal L and Tom Christiansen. Learning Perl, 2nd Edition. O’Reilly, Jul. 1997.

Stackhouse, Jim. “The C Programming Language”. 1996. URL: http://www.csc.vill.edu/~lab/C/#HISTORY (20 Apr. 2003).

Sullivan, Bob. “Remembering the **net crash of ‘88**”. 2 Nov. 1998. URL: http://www.msnbc.com/news/209745.asp (20 Apr. 2003).

TechDictionary. URL:http://techdictionary.com/Action.Lasso (14 Apr. 2003).

Warfield, Michael H. “Musings on open source security models”. URL: http://www.linuxworld.com/linuxworld/lw-1998-11/lw-11-ramparts.html (20 Apr. 2003).

Wong, David. “Secure Coding”. 20 Jun. 2002. URL: http://www.securityfocus.com/infocus/1596 (14 Apr. 2003).