Global Information Assurance Certification Paper

Stacy Ballou
GSEC Practical
Version 1.4b
September 18, 2003

MALICIOUS CODE – WHAT SHOULD WE DO?

GIAC Security Essentials Certification (GSEC)
Assignment version 1.4b, Option 1
Stacy Ballou
September 18, 2003

Stacy Ballou
GSEC Practical
Version 1.4b
September 18, 2003

MALICIOUS CODE – WHAT SHOULD WE DO?

Abstract

The information systems today are being inundated with new software packages that have potential for unexpected behavior via malicious code. The consumer and user of the products whether it is freeware or purchased software, Commercial Off the Shelf (COTS), should be provided with some type of certification that the software products being used does not infect the computer system or network. This paper will provide information and avenues for the developer of software products as well as the user of the software products to gain confidence that a software package is not likely to contain malicious code and have a minimal risk of potential vulnerabilities in a software package.

What is Malicious Code?

Malicious code is software that performs unauthorized functions causing the normal operation of an information system to be abnormal. According to SPECTRIA InfoSec Services, malicious code is defined as "software which interferes with the normal operation of a computer system" or "software, which executes without the express consent of the user."[1]

There are several types of malicious code such as viruses, worms, Trojan horses, and programming flaws. The programming flaws can be included with malicious intent or just be bad programming practices.

It is important to ensure that the software packages are free of malicious code. The government as well as commercial vendors should extend more effort in precluding malicious code in their software development practices. There are several ways that software development practices can incorporate checks for malicious code thus promoting software that can be used with an assurance that the product is free of malicious code. The government and commercial vendors should also ensure that any freeware or shareware used in the development of a product be certified to be free of malicious code. There are methods by which the freeware and shareware can be certified as being reviewed for malicious code. The user of a software product should be able to expect to trust the software.

[1] SPECTRIA InfoSec Services, "Malicious Code 101 Definitions and Background"

Types of Malicious Code

There are many types of malicious code of which the most well known types are viruses, worms, and Trojan Horses. Other types are intentional and accidental coding flaws, logic bomb, and trapdoor/backdoor.

According to Xtra's, Active Content and Malicious Code, "A virus is a computer program that copies itself from file to file and typically performs malicious or nuisance attacks on the infected computer"[2]. Upon activation of the virus, it will copy its code into one or more larger host programs. Upon execution, the virus replicates. The viruses are hard to detect as well as hard to destroy or deactivate. They spread widely and have the potential to keep infecting the environment over and over again. Viruses are relatively easy to create and are machine and/or operating system independent. The "HI" virus as described by the Network Associates is an example of "a memory resident, file infecting virus." [3] Following is a synopsis of the information provided by Network Associates at http://vil.nai.com/vil/content/v_564.htm on the HI virus.

> Once the infected file containing the HI virus is executed, the virus is activated allowing the Hi virus to become a resident of memory on a machine. As the .EXE files are executed, they become infected. The virus attaches itself to the end of the infected program adding 460 bytes to the file length. The text string "HI" is near the end of the infected program. The date and time stamp will also be updated when this virus attaches itself to the file. This virus has been known to be in existence since 1992; however, it is still a viable virus, which continues to be used today. The virus can be spread through floppy diskettes, downloads, and the network. [4]

Xtra's, Active Content and Malicious Code, also states, "A worm spreads from computer to computer. It has the ability to replicate itself by sending out large quantities of unwanted e-mails to contacts listed in a user's e-mail address book (or by other methods)"[5]. Worms can cause havoc on networks to the point of bringing them down due to the overload of e-mails. A relatively new E-Mail worm is the W32/Colevo@MM and is described by Network Associates as "a mass-mailing worm, which harvest MSN Messenger contact addresses." [6] Following is a summary of the information

---

[2] Xtra, "Active Content and Malicious Code"
[3] Network Associates, "Hi"
[4] Network Associates, "Hi"
[5] Xtra, "Active Content and Malicious Code"
[6] Network Associates, W32/Colevo@MM

that Network Associates at http://vil.nai.com/vil/content/v_100450.htm provide on the W32/Colevo@MM worm.

> W32/Colevo@MM was discovered on June 28, 2003. The worm arrives in an email message and is activated with the execution of the file. The W32/Colevo@MM launches Internet Explorer and connects to news websites, displaying the images of the Bolivian Aymara Indian leader Evo Morales. Some of the websites it connects to are: http://news.bbc.co.uk, http://membres.lycos.fr, and http://www.soc.uu.se.
>
> When the worm is run it will copy itself to the %WINDIR% (C:\WINDOWS\System32 or C:\WINNT\System32) directory using one of the following filenames:
>
> a) All Users.exe
> b) Command.exe
> c) Hot Girl. Scr
> d) Hotmailpass.exe
> e) Inf.exe
> f) Internet Download
> g) Internet File.exe
> h) Part Hard Disk.exe
> i) Shell.exe
> j) System.exe
> k) System32.exe
> l) System64.pif
> m) Temp.exe
>
> And, the worm will copy itself to the %SYSDIR% (C:\WINDOWS\SYSTEM32 or C:\WINNT\SYSTEM32) using the inf.exe, net.com, and www.microsoft .com filenames.
>
> The W32/Colevo@MM worm leaves several TCP ports (1168, 1169, 1170, 2536) open allowing a hacker to control the machine remotely. It also creates registry keys and modifies the system.ini file so that it can be loaded upon Windows start up. W32/Colevo@MM will modify some of the registry keys to allow the worm to be executed every time an associated file extension runs.[7]

As described by Xtra's, Active Content and Malicious Code, "A Trojan Horse is any program in which malicious or harmful code is contained inside of

[7] Network Associates, W32/Colevo@MM

what appears to be a harmless program"[8]. Trojan horse software is used to edit programs even registry information, delete files, set the computer as an FTP server, obtain passwords, or spy. These are typically executable files such as Back Orifice and SubSeven, which gain remote access to a victim's computer. Back Orifice allows system administrator privileges to a remote user via the Internet without any indication of its presence to the user. Back Orifice usually is installed via a component of a software installation package, as an attachment to files or programs, or just run on its own. The user installing the hidden program believes all is well and is never the wiser to the installation. As soon as Back Orifice is installed, the computer becomes easy access anytime it is connected to the Internet. Back Orifice was publicly released in 1998 and gives the remote user full access to the infected machine. Subseven is another Trojan Horse that was created to expand on the Trojan Horse Netbus. Netbus was the first Trojan that made it easy to abuse infected systems, thus Subseven is able to do all that Netbus can do and more. Subseven allows the remote user to have file control, monitoring capabilities, and network control. Subseven was discovered in May 1999.

Writing code that is insecure is considered a coding flaw. The developers and/or programmers must be knowledgeable of the security weakness of the language they are using and of security practices used in developing secure code. Writing secure code is a difficult task, and fixing bad or insecure code is not accomplished quickly and equates to an increase in cost of the product. As Kathryn Barrett describes in her overview of the book *Secure Coding: Principles and Practices*, "According to Mark G. Graff, coauthor of *Secure Coding: Principles and Practices*, to build code that repels attack, software developers must understand where vulnerabilities come from and counteract those tendencies with time-proven practices."[9]

Software developers need to understand that the introduction of buffer overflows and race conditions are coding flaws, which have security implications. Buffer overflows can result from several programming flaws, which allow the program to store data from an external source in a pre-allocated memory space that is not large enough to handle the data. Race conditions are a result of performing two or more operations at the same time. An example is reading and writing data at the same time allowing the read data to be overwritten.

Kathryn Barrett provides the following quote from Marcus J. Ranum in her overview of the book *Secure Coding: Principles and Practices* by Mark G. Graff and Kenneth R. van Wyk, "Good programmers write good code, bad programmers write bad code, but all programmers seem to write insecure

[8] Xtra, "Active Content and Malicious Code"
[9] Barrett, Kathryn, "O'Reilly Releases Secure Coding: Principles and Practices"

code, says Marcus J. Ranum, principal author of the DEC SEAL firewall, TIS Gauntlet firewall, and Network Flight Recorder Intrusion Detection System. "[10] This is a very true statement due to the fact that most software developers are not aware of the security risks involved with the coding languages they are using or the security risks they create because of their coding practices.

Security flaws can be introduced at any point in the development of a software product. It is important for the software developer to be aware of the security requirements in each of the phases of software development. Kathryn Barrett of O'Reilly states the following about the importance of software developers being security conscience throughout the development of a software product in her overview of the book *Secure Coding: Principles and Practices:*

> *Secure Coding: Principles and Practices* makes the case that developers must be vigilant throughout the entire code lifecycle:
>
> -Architecture: during this stage, applying security principles such as "least privilege" will help limit even the impact of successful attempts to subvert software.
>
> -Design: during this stage, designers must determine how programs will behave when confronted with fatally flawed input data. The book also offers advice about performing security retrofitting when you don't have the source code—ways of protecting software from being exploited even if bugs can't be fixed.
>
> -Implementation: during this stage, programmers must sanitize all program input (the character streams representing a programs' entire interface with its environment—not just the command lines and environment variables that are the focus of most security analysis).
>
> -Testing: during this stage, programs must be checked using both static code checkers and runtime testing methods—for example, the fault injection systems now available to check for the presence of such flaws as buffer overflow.
>
> -Operations: during this stage, patch updates must be installed in a timely fashion. In early 2003, sites that had diligently applied Microsoft SQL Server updates were spared the impact of the Slammer worm that did serious damage to thousands of systems.[11]

[10] Barrett, Kathryn, "O'Reilly Releases Secure Coding: Principles and Practices"
[11] Barrett, Kathryn, "O'Reilly Releases Secure Coding: Principles and Practices"

If the software developer does not incorporate security in the development of code, the coding flaws the developer makes during the development (design, implementation, testing, and operations) of a software package gives the attacker a way to gain access to the environment the software package is running in.

The dictionary definition of logic bomb is "Code surreptitiously inserted into an application or operating system that causes it to perform some destructive or security-compromising activity whenever specified conditions are met."[12] The logic bomb is simply a virus or Trojan Horse that has a timer on it. The logic bomb lays dormant until the action or criteria is met that activates the logic bomb code.

The Backdoor malicious code is also referred to as a Trapdoor and is defined as an inconspicuous way to gain access to a program, online service, or a network. The programmer developing the code for a software package, maintainers of the code, or a hacker, deliberately installs the backdoor. There are backdoors that are put in place deliberately by reputable software developers for administrative purposes; however, these backdoors can also be used for malicious purposes. Back Orifice Trojan is an example of a backdoor.

Prevention of Malicious code in shareware and COTS

To prevent malicious code in shareware or COTS software packages used in the government or commercial environment is a matter of ensuring that the software packages being used in these critical environments have been certified.

Due to the increased use of shareware and COTS with no security certification being used in the commercial and government environments, the users of this software have made it easy for malicious code to be installed in critical environments. All that the terrorist or malicious person need do is to incorporate his code into a shareware product or a developed product whether commercial or government that he has access to. If there are no preventive measures put into place, the malicious person can do just about whatever he wants to. When the software package is installed, the malicious code is available to do its job. A terrorist could easily bring down entire networks across the world. Following is one method that can be used by the government and commercial vendors in assisting to prevent the inclusion or installation of malicious code from shareware or COTS.

[12] Die.net, "Dictionary Definition: logic bomb"

Malicious Code Reviews should be conducted on all software packages whether COTS or freeware prior to being used or installed. The user of COTS or freeware should have the software packages he is considering using in his network or including in his software package reviewed by an independent entity for malicious code. A Malicious Code Review is conducted by a team of software security experts who review every line of code for possible malicious behavior and reports the possible vulnerabilities. Currently there is not a single tool that the code can be run through to magically report all the possible vulnerabilities. Consequently, this process for large software packages is time consuming and expensive. There are some available tools, which are language specific that can assist the software security expert in determining possible vulnerabilities, thus automating a small portion of the process of doing Malicious Code Reviews. The software security expert must review the whole package for vulnerabilities and ensure that all of the source code is made available for review. The software security expert should be able to build the marketed executable image with the source code provided. Building the software package from the source code will ensure that all the pieces parts are available for the review.

The following eight categories based off of the paper "Trusting Software: Malicious Code Analyses" which was co-authored by the author of this paper and found at http://www.iamsam.com/papers/milcom_malicious_code_analyses/MCAart16.htm are items the software security expert should be reviewing the code for:

1) Passwords
   - Passwords properly safeguarded.
   - Passwords being sent in the clear

2) Networking
   - Excessive access to files across the network
   - Ports being opened that do not need to be open
   - Connection to systems or software subsystems in an unsafe manner

3) File Permissions
   - File permissions being changed via code
   - Software taking ownership of files that it shouldn't
   - Software accessing publicly writeable files/buffer/directories should be evaluated to determine it's potential for malicious exploitation.

4) Minimum Privilege

- Prevent abuse of required access privileges.
- Software shall be granted the minimum privileges available to perform its function.
- Software using excessive privileges, such as Set User Id (suid). Compromise of suid root software programs may lead to excessive user privilege.
- Software providing shell access; these should be considered suspect as they may be used to obtain excessive privileges

5) Self Replicating/modifying
   - Self-replicating code across systems
   - Self-modifying code

6) Bounds and Buffer Checks
   - Perform proper bounds and parameter checking on all input data.
   - Perform checks to determine that all arguments are current and valid for system calls
   - Code that uses unbounded string copies / arguments; such code may be vulnerable to buffer overflows

7) Race Conditions
   - Software changing parameters of critical system areas prior to their execution by a concurrent process.
   - Software improperly handling user generated asynchronous interrupts
   - Software subverted by user/program generated Symbolic links

8) Other
   - Excessive use of resources
   - Software that is never executed; such code may execute under unknown circumstances / conditions, and consumes system resources
   - Software that we do not understand; such code may perform malicious functions
   - Implicit trust relationships that could induce vulnerabilities
   - Does the software meet functional security claims (if the system purports to perform passwords/logs/security, does it actually perform those functions)
   - Code that performs a malicious activity

- Software using relative pathnames inside the program should be evaluated to determine it's potential for accessing unintended files including Dynamically Linked Libraries.[13]

Following are tools that can be used by the software security expert to assist in the malicious code review for the detection and prevention of malicious code in software products that are already developed and in use. However, these tools are programming language specific and specific in the types of vulnerabilities it can detect. With this note, the software security expert reviewing code for malicious behavior must be able to perform most of the malicious code review manually using the tools, if applicable to the software package under review, as an aid.

ITS4 was developed by Cigital to automate the security source code review for software packages written in C and C++. ITS4 is a command line tool that can be run on a Unix or Windows platform and scans the code for function calls that are potentially dangerous. The C and C++ languages are known for being prone to introducing insecure code through their supporting libraries. Functions such as *gets, sprintf, strcpy, and strcat* remain in the C standard library even though these functions have potential for buffer overflow vulnerabilities. Each of these functions is found by ITS4 with a suggestion to use the more safe alternatives to these functions. The function *stat* is searched for because it is often involved with race condition problems. The vulnerability database for ITS4 is a text file that can be modified by the user at any time. So, the user of ITS4 can add, delete, or change specific items that ITS4 will search for. If the possible vulnerability is not in the database list, the vulnerability is not searched for. Currently, ITS4 vulnerability database provides 131 calls accumulated from many different sources such as the Bugtraq archives.

Secure Software created the Rough Auditing Tool for Security (RATS) and states that "RATS scanning tool provides a security analyst with a list of potential trouble spots on which to focus, along with describing the problem, and potentially suggest remedies. It also provides a relative assessment of the potential severity of each problem, to better help an auditor prioritize. This tool also performs some basic analysis to try to rule out conditions that are obviously not problems."[14] RATS scans C, C++, Perl, PHP and Python source code flagging programming vulnerabilities or errors such as buffer overflows and race conditions. This tool will not find every error and will display some code, which are not errors. RATS simply aids in the discovery

[13] Kurowsky, Jay, Ballou, Stacy, Nitzberg, Sam and Whitley, Harold, and Wood, Richard, "Trusting Software: Malicious Code Analyses"
[14] Secure Software, "Eliminating Vulnerabilities at the Source"

of possible areas of vulnerabilities. Manual inspection is still the best and recommended when using this tool.

The security expert should use these tools to assist in locating vulnerable code, then analyzing the code to determine if the suspected code is truly vulnerable. Using the tools may reduce the time to perform a manual analysis by one fourth to one third. Analyzing code for malicious behavior is a time consuming task. The current estimation of time for a software security expert to review code for malicious behavior is based off of reviewing one to two hundred lines of code per eight-hour day. There is definitely a need for a more comprehensive automated tool to assist the security expert in reviewing the source code for malicious behavior. According to the *Information Security* article, "NSA: Code Scrubbers Needed", "Daniel Wolf, NSA's director of information assurance, told a House cybersecurity subcommittee that there's a dire need for tools that can automatically examine software code for logic bombs, backdoors and sniffers."[15]

Detection and Prevention of Malicious Code During Development.

Detection and Prevention of Malicious Code during software development can be done through software code inspection, independent vulnerability assessments, and by using tools throughout the software development cycle that identify potential area of malicious behavior.

To ensure malicious code is not inserted during development, the developing company or developer should have processes in place that mandate software code inspections with an emphasis on reviewing the code for malicious behavior. A code compare, using either manual or automated means, will be performed against all modified modules. Knowledgeable software security experts who are not the original author of the code will examine all code for malicious or unintended code. The code being inspected should be controlled through a formal configuration management environment.

Independent vulnerability assessments are used to certify that a developing company has processes in place throughout the development life cycle to protect against malicious code insertion. The developing company or agency hires an independent agency that has security experts able to perform vulnerability assessments. The security experts review the developing company's process; conduct an on-site inspection, and report possible vulnerable areas with possible solutions. The vulnerability

[15] Author Unknown, "NSA: Code Scrubbers Needed"

assessments can also include a scan of the developers network to find open ports or other possible avenues of vulnerabilities in the developers network. There are several tools the security expert can use to conduct the scan; one is Nessus, which is a freeware remote security scanner tool and is known to be quite reliable.

There are also tools, which can assist the developer in detecting coding errors during the development of the software package that can be potential avenues for malicious behavior. Parasoft has two tools: Code Wizard and Insure ++ and Rational has a tool called Purify. Again these tools are language specific, which limits their usability.

Parasoft states "CodeWizard, an advanced C/C++ source code analysis tool, uses over 300 industry-respected coding guidelines to automatically identify dangerous coding constructs that compilers do not detect. CodeWizard makes it easy to create new, customized rules through the RuleWizard feature, or to suppress rules for customized analysis. When used on a daily basis, CodeWizard simplifies code reviews and make code more readable and maintainable."[16]

CodeWizard runs in Microsoft Visual C++ 6.0 or Microsoft Visual Studio.net and can be run on UNIX platforms by replacing CC with codewizard in the makefile. CodeWizard parses C and C++ code for coding patterns that match the CodeWizard ruleset and compilers do not detect. The user will get notification of any violation to the ruleset.

Parasoft's Insure ++ is an "automatic run-time application testing tool that detects elusive errors such as memory corruption, memory leaks, memory allocation errors, variable initialization errors, variable definition conflicts, pointer errors, library errors, logic errors, and algorithmic errors."[17] Insure ++ tells the user the location of the code where the leaks occur and is also capable of checking third-party libraries and functions, which do not have to be written in C or C++. Insure ++ runs in Microsoft Visual C++ 6.0 or Microsoft Visual Studio.net on a Windows 2000 or XP platform. Insure ++ can also be run on a variety of UNIX platforms. Insure ++ creates an equivalent code by inserting test and analysis functions around every line, and runs the equivalent code to determine possible problems.

Rational Purify is an automated debugging tool for Visual C++ and Java developers. Rational states that "Purify detects, locates, and diagnoses

[16] Parasoft, "Code Wizard"
[17] Parasoft, "Insure ++"

runtime and memory-related bugs"[18] (in C/C++ code) and garbage collection-related memory management issues (in Java code) "that can cause application instability and can often trigger unpredictable failures".[19] Rational Purify detects memory-related coding errors, which may or may not exhibit any symptoms in the development environment and reports the location of the problem code. Many of the problems found by Rational Purify are a result of using uninitialized memory or failing to free memory after use. Rational Purify can be used independently or as a part of the existing process. Rational Purify not only detects memory errors in source code but also can find memory errors in software libraries and components that do not have the source code available.

Automation of detecting and preventing malicious code during development is very limited at this time. The tools are language and environment dependent. Thus, there is a need for the development of malicious code analysis tools that can detect most types of malicious code during development.

Conclusion

It is possible to ensure that malicious code is kept to a minimum in our computer networked environments. There is no quick and inexpensive method to ensure that malicious code is prevented. However, the more reliable and cost effective method would be to ensure malicious code prevention during the development of the software product. This is not always feasible due to the fact that a lot of the software products being used today are freeware or shareware products, which is developed without much control. These products can be updated at will and posted for download leaving the user open to malicious behavior. In this case, the only prevention is to have security experts conduct malicious code reviews on a version of the software package to certify the product for a specific version. For a large freeware package, this can be a relatively costly endeavor. But, if a critical environment sees a need for the product, the cost of a malicious code review, may be insignificant.

Today, we are at the mercy of manual code reviews with little assistance from automated tools causing the review for malicious code to be inconsistent, time consuming, and costly. The development of automated

[18] Rational, "Manage the Complexity of Software Development Rational Suite Development Studio for Windows" and Rational, "Manage the Complexity of Software Development Rational Suite Development Studio for Unix"
[19] Rational, "Manage the Complexity of Software Development Rational Suite Development Studio for Windows" and Rational, "Manage the Complexity of Software Development Rational Suite Development Studio for Unix"

tools to assist in the discovery of malicious code is highly needed in our quick paced environment we live in today. Automation of malicious code discovery would allow software packages to be reviewed for malicious behavior in a timelier and more cost effective manner.

References

1. SPECTRIA InfoSec Services, “Malicious Code 101 Definitions and Background”, 2000, URL: http://www.securitywebsites.com/SpectriaMalaciousCode101.htm, 30 July 2003
2. Xtra, “Active Content and Malicious Code”, 3 March 2003, URL: http://www.xtra.co.nz/help/0,,4155-649452,00.html, 13 June 2003
3. Xtra, “Active Content and Malicious Code”, 3 March 2003, URL: http://www.xtra.co.nz/help/0,,5739-649452,00.html, 15 September 2003
4. Network Associates, “Hi”, 15 August 1992, URL:http://vil.nai.com/vil/content/v_564.htm, 30 June 2003
5. Network Associates, “W32/Colevo@MM”, 08 July 2003, URL:http://vil.nai.com/vil/content/v_100450.htm, 15 July 2003
6. Barrett, Kathryn, “O’Reilly Releases Secure Coding: Principles and Practices”, 30 June 2003, URL: http://www.linuxsecurity.com/articles/documentation_article7562.html , 22 July 2003
7. Die.net, “Dictionary Definition: logic bomb”, 9 February 2002, URL: http://dict.die.net/logic%20bomb/, 16 September 2003
8. Kurowsky, Jay, Ballou, Stacy, Nitzberg, Sam and Whitley, Harold, and Wood, Richard, “Trusting Software: Malicious Code Analyses”, 1999, URL: http://www.iamsam.com/papers/milcom_malicious_code_analyses/MCAart16.htm, 12 September 2003
9. Cigital, “ITS4: Software Security Tool”, 1995-2003, URL: http://www.cigital.com/its4/, 22 July 2003
10. Secure Software, “Eliminating Vulnerabilities at the Source”, 2002, URL:http://www.securesoftware.com/download_form_rats.htm, 22 July 2003
11. Nessus, “Nessus”, 02 July 2003, URL:http://www.nessus.org, 30 July 2003
12. Author Unknown, ”NSA: Code Scrubbers Needed”, Information Security, September 2003: 18
13. Parasoft, “Code Wizard”, 1996-2003, URL:http://www.parasoft.com/jsp/products/home.jsp?product=Wizard&itemId=54, 25 July 2003
14. Parasoft, “Insure ++”, 1996-2003, URL: http://www.parasoft.com/jsp/products.jsp?itemId=10, 25 July 2003
15. Rational, “Manage the Complexity of Software Development Rational Suite Development Studio for Windows”, 2000, URL: http://www.unicorn.cz/distribution/_downloads/ds-developmentstudio.pdf , 25 July 2003

16. Rational, “Manage the Complexity of Software Development Rational Suite Development Studio for Unix”, 2000, URL: http://www.rational.com/media/products/dstudio/D805_DevStudioUnix.pdf, 25 July 2003