Global Information Assurance Certification Paper

Short, Christopher
GSEC 1.2f

SOURCE CODE REVELATION VULNERABILITIES

Introduction

Many security related articles and papers concern protecting networks and servers from outside intrusion, and rightly so. This is understandable since many security professionals do not hail from a programming background. However, application security cannot be ignored in today's complex and competitive environment.

The most commonly discussed aspect of application security concerns protecting an application from deliberate misuse such as buffer overruns and other such vulnerabilities. These can often be discovered or exploited without access to source code. However, protecting source code is a vital part of information security. Without debating the merits of open vs. closed source programming and business models, it is fair to say that not everyone wishes to share their source code with the world, either because it contains trade secrets or because the author or publisher chooses not to have code examined.

Traditional Vulnerabilities to Reverse Engineering

As in other areas of security, a belief in the invulnerability "security through obscurity" is often misplaced. Most software (open-source excepted) has been traditionally distributed as executable files, which of course are not human-readable. However, technologies have existed for some time to reverse-engineer such code.

Disassemblers are tools that can turn an executable file into human-readable code. The chief problem with disassemblers has been that they turn executable files into Assembly language files. The ability to read Assembly language, which is low level code, is not common, as most programmers code in higher-level languages. A secondary problem with this is that the program in question probably did not originate as Assembly language, but rather as C, Java, Visual Basic, or any of many other languages.

A less mature and therefore less reliable technology is the decompiler. Conceptually similar to a disassembler, the decompiler is designed to translate an executable file into a higher-level language, generally C. An obvious problem with this is that decompilers are not precise—there are often many ways to achieve the same result when writing software, and it is probable that there will be differences between decompiled code and the original source. Of course, since a programmer's comments are not included in a compiled executable, these will also not be available in decompiled source code. (McGraw and Viega)

Still, the vulnerabilities here should be obvious. Most programmers either know C or can learn to read it reasonably quickly, and common vulnerabilities such as hard-coded passwords and such will be immediately obvious.

Vulnerabilities in Virtual-Machine Environments

The Java environment has obviously become very prolific in recent years, with .NET from Microsoft poised to do so as well. Both environments use virtual machines, in which code is not compiled to machine language, but is instead compiled to a form useable on multiple machine types. (This is referred to as "bytecode" in Java, and "IL" in .NET.)

Since virtual machine code is designed to run on multiple processors via a virtual machine, it follows that the "compiled" code would be simpler, and therefore easier to understand. (Travis, Greg) In most compiled languages, information contained in source code (other than literal values) is not included in the compiled executable. With Java, a significant amount of source code information is included in the bytecode. This can facilitate decompilation, and since bytecode generally only comes from Java. (Low)

One of the first Java decompilers was a product called Mocha, written by Hanpeter van Vliet. A complete set of examples of how Mocha decompiles code, and how the decomplied version differs from the original, may be found at http://www-106.ibm.com/developerworks/java/library/j-obfus/?loc=tstheme. Mr. Van Vliet later came up with Crema, which is a "code obfuscator" designed to confuse decompilers. It has a particularly curious ability to create obfuscated code that will cause Mocha to crash. (Travis)

Microsoft's .NET platform has not even been commercially released yet, and already security concerns are mounting. Just as programmers are often not concerned about security issues while writing code, it may also be the case that some will prove unconcerned with the impact of methodology on security.

Because of the way Microsoft's IL works, applications compiled in anything other than "unmanaged" C++ will not be secure when written as a desktop application. (Fergus). This type of application is the traditional Windows program that so many programmers are familiar with. One danger is that programmers who develop desktop applications in the current development environment will do so in .NET. Just as Java applications are deployed as bytecode, .NET applications must be deployed as IL, which is very easily reverse-engineered. In fact, Microsoft provides a tool (ILDASM) which provides well-formatted IL code complete with metadata that describes what a given application is doing.

The "defensive programming" tactic to combat this vulnerability is to use either unmanged (non-IL) code, or to have all valuable code run on servers and expose only interfaces, so there is no chance of anyone seeing the IL code. This is Microsoft's preferred method. Since adhering to this method also promotes their services vision of software, it would be a reasonable guess that this situation will not change in the near future.

The impact of this vulnerability is twofold: it may serve to limit third-party development of

applications for distribution to end users, and it may also serve to make vendors suspicious about selling server-based software. After all, the number of people with administrative access to servers is not small, and vendors will not be willing to trust every administrator's honesty. (Low) This kind of environment would seem to violate the principle of security as an enabler. There is almost nothing a security administrator can to do address this weakness, other than educate developers and management about it. The potential for theft of intellectual property is enormous, and the responsibility of network and server administrators to prevent access to server-based code will be grave indeed.

Vulnerabilities in Scripting Languages

Reverse engineering takes on a slightly different form when dealing with scripting languages, particularly those used on web sites. Since scripting languages are by definition not compiled, there is nothing to reverse engineer, and the source code revealed is the actual production code, not a "best effort" version which may or may not look like the original.

One fairly well-known and financially damaging vulnerability of client-side code is validation routines, or perhaps a lack of them in the proper place. Any code sent, especially in clear text, to a device over the Internet should be considered suspect once it leaves an organization. Some electronic shopping cart programs make use of "hidden fields" to echo pricing data to a web browser, usually in the absence of another method of maintaining state. Others rely solely on client-side validation scripts to prevent pollution of databases, and possibly buffer overrun attacks due to invalid input. The danger in relying on anything sent to the client is that the user can save the page with the code, alter it, and return something unexpected. Additionally, having validation routines in clear text can tell an attacker the kind of input that a server cannot handle.

Unfortunately, not using client side scripting is generally not an option. Such programs can save multiple round-trips to a server, and can catch data errors earlier. Clearly they have their place. However, most security-conscious web experts recommend always validating everything on the server even if client-side validation is available. This results in duplicate validation and duplicate code, but it delivers the maximum assurance of good data input. What it does not do, however, is hide any business logic that may be present in client side code. For this reason, client side scripting languages should be limited to common things like date and character validation. Proprietary routines and specialized code should be left on the server.

Server scripts present their own problems, though. While only the output of server scripts is supposed to be echoed to the client, serious vulnerabilities occur because very sensitive things such as database passwords are often coded into web server scripts (McClure and Scambray) These things occur because some web servers contain vulnerabilities which permit an attacker to access the actual code. Even a normally secure server can be vulnerable to this if it is misconfigured.

Microsoft's Internet Information Server has in the past had some rather serious source code revelation bugs in it. Version 3 of IIS had a vulnerability in which would transmit source code if one appended one or more dots to the end of an ASP URL. This vulnerability was patched, and

IIS3 is not used frequently these days. However, IIS version 4 is still widely used, and it has a similar vulnerability. By appending "::DATA", one can download the source code from an ASP page on an unpatched version of IIS4. (McClure, Scambray, and Kurtz)

Another vulnerability which IIS systems should be patched against is some sample scripts that were included in IIS 4.0. These scripts (showcode.asp, code.asp, and codwbrws.asp) are designed to display source files over the web. They were intended as a tool to assist developers. Unfortunately, due to a programming flaw, the use of the ".." character sequence was not restricted. (McClure, Scambray, and Kurtz) In Windows as in Unix, this refers to the directory one level above the current one. If the IIS user account has read privileges on the directory in question, any file can be displayed over the web, including source code, system configuration, etc. A patch does exist for this vulnerability. Still, one of the best recommendations is to not install sample files, or anything else not used actively, on any production server.

While script files provide an easy way to program web servers, they do not do much for code reuse or object-oriented programming. To partially compensate for this, ASP programmers often use "include" files, which are simply script files that can be dynamically inserted into an ASP page through the use of a special tag. The vulnerability here is that include files, which typically have a .INC extension, are rendered differently than ASP pages. ASP pages are interpreted and (depending on the server version) compiled by the web server, which is why the source code is not normally visible to a user browsing to the page. However, .INC files by themselves are sent to browsers as clear text if an attacker knows the name of the file in question. No patch exists for this—the suggested remedy is to rename the files such that attackers will have a harder time finding them, since the .INC extension is not a requirement for an "include" file.

One would think that Microsoft would have addressed source code vulnerabilities in IIS5, which ships with Windows 2000. Unfortunately, this has not been the case. By "sending a special header with 'Translate: f' at the end of it, and then a trailing slash '/' appended to the end of the URL" (SecuriTeam), one can obtain the source of any script file that the user could execute. Unfortunately, this vulnerability actually exists in the FrontPage 2000 Server Extensions, the problem can be ported to IIS 4 as well. This particular vulnerability was caused by Microsoft's implementation of WebDAV, which is a W3C standard method of implementing authoring on web sites.

Lest one get the idea that this sort of thing only happens to Microsoft, in January, 2001, a vulnerability was discovered in the popular PHP scripting language, commonly used with the Apache web server. This vulnerability occurs when an installed version of PHP is shut down and inadvertently shuts down other virtual machines. Since the code is no longer being interpreted, it is downloadable by external users who would normally only execute it.

Recommendations

As with anything else, protecting an organization's assets, including its source code, begins and ends with education. Many programmers are simply ignorant of the vulnerabilities inherent in

certain architectures and how to avoid them. In addition, non-technical management cannot be reasonably expected to know that their perfectly functioning software might be revealing more than they realize. Even code without proprietary secrets can be used as part of an intelligence-gathering operation. Management should be expected to know the value of the information stored on systems, however, and they need to be reminded or informed of just what is at risk.

This would be yet another excellent opportunity to take a look at relevant security policies. If an organization doesn't have one, it should set about developing one. If the security policy does not address programming concerns and asset protection, these things should be added.

Since many web site source code revelation problems can eventually be fixed with patches to the web server, administrators should always follow the time-honored advice of making certain that their servers have up to date patches installed. Those vulnerabilities that cannot be fixed by server patches need to be fixed with solid design and coding practices, and a security review of all production systems architectures. A prohibition against including proprietary code in client-side scripting would be a good start.

Protecting from reverse engineering is not an easy task, but there are things organizations can do to make life difficult for reverse-engineers. Code can be encrypted, but unless specialized hardware is used, the encryption keys would have to be present with the code, and thus vulnerable themselves (Low) While many might say that programmers naturally obfuscate code anyway, these products are designed to perform some interesting operations. Changing names and identifiers in code can confuse readers. Changing data storage and counting (for instance, making a loop count from 331 to 340 instead of from 1 to 10) can also hide a program's functionality. Finally, the flow of a program can be altered by such things as including decoy procedures that never execute, having loops count backwards, etc. (Low)

One rather obvious problem with inserting extra code and deliberate nonsense into a running program is that something has to sort all of this out when the program is run. It is possible that code obfuscators can cause additional consumption of system resources. However, in the case of virtual machine languages like Java and anything running under .NET, the use of a code obfuscator should be made mandatory for any compiled code which is distributed to the outside world.

Another obvious solution to the decompiling problem is to not release code to the outside world in the first place. It is curious that the virtual machine languages, which are so vulnerable to reverse engineering, are the very things being used to promote web services and server-based programming. While these techniques have their own sets of problems, if a user cannot see the compiled code, it stands to reason that the code cannot be decompiled.

It is in this area that good network and server administration is essential. Server-based programs are rather likely to have access to sensitive information and to need privileges to system resources that malicious hackers covet. If an attacker can gain access to the source code of these programs, he or she can easily blueprint internal system architecture. This cannot be stressed enough: even if the source code has no intrinsic value, it can still provide knowledge necessary for a successful

attack.

There are things system architects can do to minimize vulnerabilities. Since scripts are human-readable without a decompiler, sensitive code should be compiled with more traditional tools whenever this is possible. Database access codes should never be stored in scripts—they should be read from a highly controlled area on the server.

While those things deal with program architecture, some things are left to the more traditional areas of security administration. Servers should be hardened to the extent possible. Particular attention must be paid to making certain that external users cannot access script source code from web servers.

One of the long-standing stereotypes of the IT world is that programmers and administrators barely tolerate each other. While it is almost certain that a number of programmers don't care what happens to their code once they compile it, and that security and network people frequently don't care about programming techniques. Source code revelation is a cross-disciplinary problem. Cross-disciplinary problems require good management and good communication. It must be remembered that malicious attackers are themselves multi-disciplinary creatures. The vulnerabilities discussed here and others like them must not be ignored simply because they don't fit neatly into one side of the IT world or the other.

References

Fergus, Dan "Is Your Code Safe?" May 2001.
URL: http://www.vbpj.com/upload/free/features/vbpj/2001/05may01/df0501/df0501p.asp (29 August 2001).

"ISS E-Security Alert: Form Tampering Vulnerabilities in Several Web-Based Shopping Cart." 1 February 2000.
URL: http://www.safermag.com/html/safer22/advisories/24.html(29 August 2001).

Low, Douglas. "Protecting Java Code via Code Obfuscation." 25 January 2001.
URL: http://www.acm.org/crossroads/xrds4-3/codeob.html (13 August 2001).

McClure, Stuart, Scambray, Joel, and Kurtz, George. "Hacking Exposed." Osborne/McGraw-Hill 1999. 413-414.

McClure, Stuart and Scambray, Joel. "Protecting IIS systems requires hiding your Web-script source code from prying eyes." Infoworld, Security Advisor.
URL: http://www.infoworld.com/articles/op/xml/00/10/30/001030opswatch.xml (29 August 2001).

McGraw, Gary and Felten, Edward "Twelve rules for developing more secure Java code."
URL: http://www.javaworld.com/javaworld/jw-12-1998/jw-12-securityrules.html (29 August 2001).

McGraw, Gary and Viega, John "Make your software behave: Security by Obscurity." IBM DeveloperWorks. October 2000.
URL: http://www-106.ibm.com/developerworks/java/library/s-obs.html (9 August 2001)

Mehta, Manish B. "Protecting IL Code from Unauthorised Disassembling." 19 April 2001.
URL: http://www.c-sharpcorner.com/Security/ProtectILCode.asp(16 August 2001).

"PHP Engine Disable Source Viewing Vulnerability." 19 January 2001.
URL: http://www.safermag.com/html/safer33/alerts/52.html(28 August 2001).

Robbins, John "ILDASM is Your New Best Friend" Bugslayer. May 2001.
URL: http://msdn.microsoft.com/msdnmag/issues/01/05/bugslayer/bugslayer0105.asp(28 August 2001).

“Translate: f vulnerability exposes IIS files source”. 21 August 2000.
URL: http://www.securiteam.com/windowsntfocus/5VP0P0A2AS.html (29 August 2001).

Travis, Greg. "How to lock down your Java code (or open up someone else's)." May 2001.
URL: http://www-106.ibm.com/developerworks/java/library/j-obfus/?loc=tstheme(29 August 2001).