Code Injection in Windows Systems:

Where do We Stand Now?

Nick Sempere
Advisor: Ming Chow

Abstract

Since the mass production and distribution of personal computers began, their operating systems have made huge strides in terms of computational abilities and sophistication. With all of the added fortitude, however, these operating system cannot seem to shake their one true achilles heel: security. Despite no small amount of effort, developers seem to be in a constant cycle, identifying and patching vulnerabilities only to encounter more with the next release. This cycle is particularly evident in Windows operating systems, which represents a huge portion of the computers currently in use by the public. With the state of security in Windows systems in such an apparent state of anarchy, it should be no wonder that mainstream users find themselves in a world of uncertainty. What should they be worried about? What should they no longer have to worry about? Where does the greatest risk lie? This paper seeks to answer those questions as it explores the current state of security in Windows operating systems with a focus on one breed of vulnerabilities in particular: code injection.

To The Community

Simply put, there are a myriad of ways to maliciously inject code nowadays. That is scary because injection acts as one of the most popular vehicles through which malwares establish a foothold in a system or network. While practically every major operating system shares this common threat, Windows systems and their vulnerabilities struck me as most worthy of further research, as they still represent the majority of system that are in use today. I chose this topic in an attempt to make the current state of injection techniques as clear as possible and point to the newest or most ill-addressed methods that attackers are using against Windows systems now.

Introduction

Code injection is a malicious technique that involves tampering with a program or application in a way that causes it to execute commands that it was not meant to execute. An attacker can employ injection to damage an application's infrastructure, steal sensitive information, or complete any number of harmful tasks. While injections can be made to target SQL queries or execute javascript code, attackers can also send injections deeper, targeting a machine's operating system directly. In what ways are modern operating systems more vulnerable to code injection? This discussion aims to explore techniques such as command injection, DLL injection, and Powershell injection in one particular mainstream line of operating systems: Windows. Through my research, I argue that although command injection and DLL injection pose relatively lower threats to modern Windows systems, powershell injection is far more formidable, and that today's users must not overlook it.

Command Injection

A command injection is a vulnerability in which an application fails to properly escape special characters that could mutate or extend the OS commands it sends to its backend [1]. Attackers can use this vulnerability to execute arbitrary shell code on a target machine through a web application or any other potential remote entry point. This is one of the most commonly-cited injection vulnerabilities still in existence because it is so easily overlooked. Command injection occurs at an application level, built on top of the Windows operating system itself. That means that developers of those applications, rather than the developers of Windows, are responsible for preventing it. Because these applications come from so many different organizations (be they corporations not affiliated with Microsoft, or independent developers) however, it is extraordinarily difficult to ensure code standards that prevent command injection.

This weakness can make its presence felt in two particularly common ways. The first such exploit involves the abuse of user input fields in an application. If the application's source code fails to ensure that input text is void of special characters such as ‘ / ’ and ‘ ; ’, it will blindly evaluate that input. If an attacker attaches additional commands, the operating system will evaluate those too. Those added commands could prove disastrous to the application's infrastructure, and maybe even to the system as a whole.

A second common implementation flaw relates the improper referencing of environment variables in scripts associated with Windows applications [2]. These environment variables can be referenced in two ways: with surrounding quotations and without them. When they are referenced without quotations, the operating system evaluates the variable itself, along with any commands that it points to. Attackers can exploit this referencing style by modifying one of these variables, tying its evaluation to other arbitrary commands [3]

Risk Assessment: Low

While the potential for a successful command injection of any form to cause damage is very high, today's users should, in general, feel less threatened by the vulnerability. Firstly, injection points opened by an improperly referenced environment variable in a Windows script is less accessible than similar flaws in linux/unix systems [2]. Because Windows shells do not interact with the command line in the same way that Linux Bash does, *remote* execution of such an attack is impractical, if at all possible in the first place. Without the threat of attackers exploiting it remotely, the vulnerability itself loses a considerable amount of bite. In fact, regardless of its restricted availability, the environment variable evaluation vulnerability should be the least of one's concerns if exploited. Creating an environment variable or directory is an action that requires elevated privileges. A code injection made via an environment variable would imply that the attacker already has some higher level of privileges; that is a far more troubling problem [2].

Another fact that modern Windows users and developers should take solace in is the fact that both flaws are easily-mended. A direct command injection can be thwarted by simply sanitizing user input, while injection through the evaluation of an environment variable is a similarly trivial fix; by referencing environment variables in quotes [2], the operating system will not evaluate the name or any code associated with that name.

DLL Injection

DLL injections are an extremely common technique and are often used benignly. In fact, Windows systems are designed to incorporate them; DLLs, or dynamic link libraries, encourage more effient use of resources and help promote modular software design [4]. This design choice, however, can be exploited if an attacker were to injection his or her own maliciously engineering link library.

A DLL can be hooked and executed in many ways, such as bulldozing the address of another pre-existing method or creating an entirely new process from which to link the file. One of the least detectable, and by extension, the most worrisome methods involves injecting a DLL into a process that is already running [5]. There exist a number of methods that Windows uses to import DLLs into live processes. Take `AppInit_DLLs`, for example. `AppInit_DLLs` accepts a list of DLLs and loads each of them into all the currently-running user mode processes [6]. Since the release of Windows 7, however, `AppInit_DLLs` has enforced code signing on all DLLs passed through it [6]. This modification makes the injection of unauthorized DLL's practically impossible.

There exist other injection methods, however. Though more obtrusive, it is possible to bulldoze the address of a function that already exists with the address of some malicious DLL [6]. If an attacker were to successfully tamper with a process's memory in that way, the consequences could be disastrous.

Risk Assessment: Low to Moderate

As with command injections through variable evaluation, DLL injection requires root privileges. Thusly, an attacker would have had to exploit one or more vulnerabilities before reaching a stage where injection is possible [5]. On the other hand, a malicious payload could arrive at such a point quickly if it was delivered through social engineering; phishing still remains one of the most generally successful attacks around. On a reassuring note, as was mentioned earlier, a number of injection methods such as `AppInit_DLLs` now require code-signing. Replicating these signatures simply isn't feasible, meaning that Windows has successfully closed a number of doors that were previously ajar.

The fact remains, however, that Windows was designed to allow DLL linking and the ease-of-use that this ability entails is a double-edged sword. Development is easier, but so is malicious injection. That being said, this vulnerability cannot be ignored by any Windows user who take the security of his or her system seriously.

PowerShell

Powershell is a tool developed by Microsoft that is comprised of a scripting language command-line shell. It is a powerful tool for system administration and has been standard to all new Windows systems since Windows 7 [7]. Because it makes system administration and other software management so much easier, it has become a very popular tool and a number of software packages, such as ManageEngine's OpManager, have been developed to centralize these procedures [8]. A common feature of such packages is some sort of system command input. Were an attacker to gain access to a tool like this, he or she could use the input line to string together multiple commands in a similar fashion to a standard command injection made over the web. Much like shell scripts on Linux distributions, Powershell uses the ‘ ; ’ character to separate commands [9].

Risk Assessment: Moderate to High

At a first glance, it is tempting to surmise that this vulnerability is one in the same with command injection and that it should therefore be regarded with the same degree of attention. In many senses, the two flaws are built on the exact same premise. However, a number of factors make Powershell stand out. Firstly, these system management packages are designed assuming that the users they allow to execute commands are authorized to do so. That means that they are less likely to sanitize command inputs. Should an attacker establish access to one of these systems, say, through targeted social engineering, he or she would be home free.

Most notably, however, is the fact that Powershell itself, is simply quite a new technology. Users have enough had time to see the benefits of using it, but are still at the stage where they have yet to learn about its potential vulnerabilities. Such is the cycle of technology. Though fixing these problem is relatively uncomplicated, the novelty of the technology surrounding it has the potential to open doors. It is absolutely imperative that we both recognize and respect that novelty.

Conclucion

It is distinctly possible that code injection is a vulnerability that operating systems like Windows will never be able to completely eliminate. There is no time to speculate on the future however. The only relevant fact is that these vulnerabilities exist at the moment and they inflict profound, lasting damage to important systems and applications on an almost daily basis. In a landscape where new technologies roll in and out of the spotlight so regularly, the most pragmatic step that today's users can take is to pause and take a look at where security is now and where it is going. I believe that we are starting to see meaningful progress in the prevention of attacks like command injection and DLL injection and that now, injection through powershell poses a much newer threat.

Works Cited

[1] Common Weakness Enumeration entry No. 78: https://cwe.mitre.org/data/definitions/78.html

[2] Michael Mimoso: “Shellshock-like Weakness May Affect Windows”. 6 October 2014 https://threatpost.com/shellshock-like-weakness-may-affect-windows/108696/

[3] The Security Factory: “Command-Injection Vulnerability for Command-Shell Scripts. https://www.thesecurityfactory.be/command-injection-windows.html

[4] Microsoft Corporation: “What is a DLL?”. https://support.microsoft.com/en-us/kb/815065

[5] Dejan Lukan: “API Hooking and DLL Injection on Windows”. 31 May 2015 http://resources.infosecinstitute.com/api-hooking-and-dll-injection-on-windows/

[6] Microsoft Corporation (Windows Dev Center): “AppInit_DLLs in Windows 7 and Windows Server 2008 R2”. https://msdn.microsoft.com/en-us/library/windows/desktop/dd744762 (v=vs.85).aspx

[7]Eduard Kovacs: “Windows Powershell Increasingly Abused by Attackers” 2 June 2014 http://www.securityweek.com/windows-powershell-increasingly-abused-attackers

[8]Chris (Obscuresecurity): “Command Injection to Code Execution with Powershell”. 27 April 2012 http://obscuresecurity.blogspot.com/2012/04/command-injection-to-code-execution.html

[9] Jeffery Snover (Windows Powershell Blog): “Protecting Against Malicious Code Injection”. 22 November 2006. http://blogs.msdn.com/b/powershell/archive/2006/11/23/protecting-against-malicious-code-injection.aspx