

Code-Driven Attacks: The Evolving Internet Threat*

Anup K. Ghosh

Cigital

21351 Ridgetop Circle, #400, Dulles, VA 20166

phone: (703) 404-9293, fax: (703) 404-9295

email: aghosh@cigital.com

www.cigital.com

Keywords: active scripting, code-driven attacks, malicious code, mobile code, viruses, worms.

Abstract

Intrusion detection research over the last twenty years has focused on the threat of individuals illegally hacking into systems. Over the last two years, the intrusion threat to computer systems has changed radically. Instead of dealing with hackers, we are now faced with defending our systems against code-driven attacks. Similarly, the anti-virus community has failed to respond adequately to viruses that have evolved from residing in floppies and documents to Internet savvy code that leverage Internet connections to spread. Today's anti-virus tools are not sufficiently prepared to address the Internet-enabled threat. Whereas intrusion detection tools are more naturally positioned to address an Internet threat, the nature of the code-driven threat is so different from human hacking as to render most intrusion detection approaches ineffective. One of the key characteristics of this new threat is that malicious code can spread on an Internet scale in Internet time, rendering current intrusion detection approaches impotent. Thus, the two primary defenses we have — intrusion detection tools and anti-virus tools — are largely ineffective in dealing with code-driven attacks. In this short paper, we describe the code-driven attack threat, then briefly describe an approach to addressing a specific class of this threat — active-scripting-based attacks.

1 Introduction

The U.S. has invested significantly in intrusion detection research over the last 20 years. The results are fairly significant — most notably a thriving intrusion detection commercial industry as well as a number of different intrusion detection research programs. While the results of these efforts are well-founded on solid research, the nature of the threat against computer systems is changing radically and calls for a re-thinking of the type of security research necessary to counter the evolving threat.

Whereas intrusion detection tools and research are largely concerned with identifying and stopping hackers or individuals attempting to break into systems, the most significant emerging intrusion threat is code-driven attacks. A code-driven attack will typically use an Internet service to send malicious code to end users, who then inadvertently execute the code. Mobile code will download and sometimes run transparently to the user, while other times, the user is fooled into running the code by social engineering tricks, such as tricking the user into believing the code is a legitimate document sent by an acquaintance.

Code-driven attacks use a variety of different software technologies including executable attachments to email such as binaries and scripts, embedded scripts in HTML mail, Web-based mobile code such as Java applets, ActiveX controls, Javascript, Visual Basic scripts, and Visual Basic Application (VBA) macros used to drive applications. In addition, Internet chat rooms and newsgroups often serve as an initial dissemination post for malicious code. Finally, code-driven attacks also use more traditional forms of transmission such

*This work is sponsored by the Defense Advanced Research Projects Agency (DARPA) under Contracts F30602-99-C-0172 and DAAH01-98-C-R145. THE VIEWS AND CONCLUSIONS CONTAINED IN THIS DOCUMENT ARE THOSE OF THE AUTHOR AND SHOULD NOT BE INTERPRETED AS REPRESENTING THE OFFICIAL POLICIES, EITHER EXPRESSED OR IMPLIED, OF THE DEFENSE ADVANCED RESEARCH PROJECTS AGENCY OR THE U.S. GOVERNMENT.

as FTP services as well as physical media to spread among computers. It is important that our defenses address code-driven attacks regardless of the entry mechanism.

Code-driven attacks find themselves bridging research communities of interest with different origins. For instance, the anti-virus research community has long been tasked with addressing the malicious software (or malware) problem. However, today's most virulent viruses use the Internet to spread on a scale and time frame unsuitable to current signature-based anti-virus detection tools. Likewise, because intrusion detection tools have traditionally focused on the Internet as the vector of most attacks against computing systems, the intrusion detection community would seem like a more natural fit for addressing the problems of code-driven attacks. But again, the scale and time frame at which code-driven attacks spread are unsuitable for most current intrusion detection tools which are mostly oriented toward detecting humans hacking into systems — a far cry from the global nature and compressed time scale of Internet-enabled code-driven attacks.

In this paper, we argue why current approaches in these areas are inadequate for the code-driven attack problem and the future emerging scope of these threats. Then, we describe our approach to addressing current and future threats from a significant class of code-driven attacks: active-scripting-based attacks.

2 The Problem of Code Driven Attacks

As stated earlier, the most significant threat of intrusions is rapidly changing from individual-based hack-in attempts to code-driven attacks. However, this trend does not mean that individuals will not hack into systems, or that we should let down our defenses against individual attacks. Rather, work should and will continue to detect hackers breaking into systems. In fact, the commercial sector has responded with a number of intrusion detection tools that have incorporated largely signature-based detection tools.

However, more recent trends indicate that the most costly intrusions are being perpetrated by code, rather than individual hack-in attempts. For instance, the Melissa virus was the precursor to a number of VBScript-based attacks. More recently, the Love Letter virus is estimated to have cost businesses \$8 billion in lost productivity. Some of these attacks are particularly pernicious. For instance, some VBScript attacks will erase all files of a particular type, such as *.c, *.cpp, *.doc, *.ppt files in an irrecoverable manner unless backed-up off disk. The Love Letter virus replaced .jpg files with its own code such that in the future, if the user decides to open a .jpg image file, instead of displaying the intended image, the user actually ends up launching the virus and mailing it out to a number of people on their contacts address book.

This last feature of viruses, or more accurately, worms, is what has made this class of malware particularly damaging. Code that not only damages the user's system, but also spreads itself over Internet connections, can quickly bring down very large segments of the Internet, and by extension, the business world. Today's virus and intrusion detection tools are ineffective against countering a threat that can rapidly spread in a matter of hours instead of days. Thus, we must build defenses against not only damage left by malware, but also against malware proliferation.

Mobile code has become not only pervasive on Web pages, but also a essential to Web-based systems. It is difficult, for instance, to browse the Web with Javascript disabled, because Web development has become so heavily dependent on running Javascript. While Java applets were originally considered the most common example of mobile code, the reality is Web-based scripts such as Javascript, JScript, and VBscripts are much more pervasive. Unfortunately, the security research community has paid little attention to the security concerns with running such code, despite the overwhelming dangers as illustrated by the Melissa-style VBscript viruses.

Another class of mobile code that presents a real danger is email attachments. Once received, the user will typically run the attachment by simply double-clicking on it. Most security specialists believe that user education (*i.e.*, instructing users not to run unknown attachments) is the solution to this problem. However, as these viruses illustrate time and again, user education does not provide enough assurance. Social engineering is typically used to convince users that the attachment is really of interest. For instance, they are unintentionally sent by someone they know and instructed to review the attachment or view an image file that really is not an image.

These forms of code-driven attacks are really the new types of intrusions that we need to address. They enter systems unimpeded by firewalls. They go past intrusion detection systems undetected. They propagate too fast for anti-virus vendors to disseminate their signatures in time. They can damage our file systems and

send sensitive information out over network sockets. They can even spy on our computer and Web usage patterns.

Today's approaches to dealing with these forms of malware are not effective. The most common approach to dealing with VBScript viruses is to use a signature-based virus detection tool. Unfortunately, this approach will not detect the next variation of the same attack. Because these viruses spread so quickly, the reactive approach to virus detection is not effective in stopping a Melissa-style virus. Another approach to stopping Active Scripting attacks is to disable active scripting within Web browsers and mailers. However, this is not a good solution, either, because active scripting is used pervasively in Web pages. In addition, many active scripting attacks run from the file system and will control other desktop applications. Simply shutting off active scripting capabilities in browsers and mailers will not adequately address VBScripts, Javascripts, and other scripting technologies (*e.g.*, Python, Rexx, Perl) that run directly on the host platform interpreter. Other approaches such as filtering content at the firewall have proved to be largely ineffective in the past and will be rendered obsolete in the future by end-to-end encryption.

3 Next Generation Code-Driven Attacks

It has been often said that we dodged a bullet with Melissa. That is, the damage from the Melissa virus could have been much more severe. The truth is the damage from all of the scripting-based attacks we have seen in the wild has been relatively tame in comparison with the possible damage they could inflict if so programmed. In spite of the relatively tame nature of these viruses, the cost of dealing with them and the losses in productivity and sales is estimated to exceed \$266 billion in the US in the year 2000 alone.¹

Given that this number represents more than 2.5% of the US Gross Domestic Product, analysts and investors on Wall Street and civil and military personnel in the US Government have begun to realize that malicious code threatens our very own information-based "new" economy. And yet, we have still dodged a bullet.

Michal Zalewski wrote an interesting essay/cookbook titled "I don't think I really love you: or writing internet worms for fun and profit" which describes a project that he and some of his colleagues have been working on since 1998 called "Samhain"². The project involves building a prototype worm that is capable of:

- being architecture-independent, *i.e.*, portable across different operating systems,
- invisible to its victims using process hiding facilities,
- autonomous, so that it can independently migrate from host to host,
- intelligent, so that it can learn new exploits and functionality,
- resilient to detection, network tracing, and reverse engineering,
- polymorphic to avoid signature detection,
- programmable, so that it can benignly infect a host, learn about its environment, then maliciously steal its secrets and cause maximum damage.

The essay describes the methods for implementing each of these desired properties of a worm, including source code. The worm has been prototyped in an "engine" for the purposes of experimentation. However, the truth is, there is no rocket science to the work described in the essay. A motivated team could design and develop a worm with these properties without too much difficulty. Such a worm is far more malicious than the current breed of Internet worms that even our current detection and anti-viral systems cannot handle.

¹Global survey of 4,900 information technology professionals across 30 nations conducted by InformationWeek Research and fielded by PricewaterhouseCoopers LLP, released July 10, 2000.

²Available online at <http://lcamtuf.na.export.pl/worm.txt>.

4 Addressing the Active Scripting Threat

The code-driven threat has largely reared its ugly head against the Windows 32-bit (Win32) platform. It is not that the Win32 platform is inherently more vulnerable than other platforms (though, some may argue that it is) as much as the ubiquity of the platform in both homes and offices makes it a good target.

One class of code-driven threats that has proven particularly popular, powerful, and troublesome is active-scripting-based attacks. For instance, in August, 2000, four of the top six malicious code threats on Symantec's Antiviral Research Center's Web page³ were active-scripting-based attacks, including: VBS.Stages.A, Wscript.KakWorm, VBS.LoveLetter, and VBS.Network.

Active-scripting-based attacks are developed in code in one of several popular scripting languages (VBScript, JScript, VBA macros, Perl, Python) that run on their respective interpreters. The interpreters reside in various application hosts including Web browsers, mailers, MS Office applications, and the Windows scripting host.

On the Win32 platform, one feature all these interpreters have in common is that they use the Windows Active Scripting interface — a set of well-documented APIs for accessing application and system resources. Active scripts use this interface to access application objects and methods, system resources, as well as to drive other applications. We leverage this interface in order to address this class of code-driven threats.

The goal of our research is to protect the Win32 platform against current, future, and unknown active-scripting-based attacks. Our approach is to effectively constrain the active scripting capability on the platform to allow benign uses while denying malicious uses. We believe this is feasible because the range of possible behaviors for active scripting functionality is much greater than the range of legitimate functionality currently used for Internet-based content.

We have implemented a prototype system that automatically wraps all calls to the Active Scripting API from all programs that run on a Win32 platform. From this layer, we can write policies at the appropriate interface level, such as the document object model (DOM) of an application, to control the way in which a script accesses an applications objects and methods. We also have control over the way a script attempts to access system resources such as the file system and network sockets. Our approach is effective against the following types of code-driven attacks: email attachments, embedded scripts in HTML mail, scripts that exploit browser holes, scripts that exploit ActiveX controls marked safe for scripting, scripting of MS Office applications, VBA document macros, and other future scripting technologies that leverage the Active Scripting API.

The key benefit of our approach is that it is not signature-based. Therefore, it handles a class of malicious code — active-scripting-based attacks — rather than specific instances. As a result, it addresses future and unknown active scripting attacks. We have demonstrated its capability to detect and prevent scripting-based attacks in emails and from Web pages. However, it is equally powerful against scripts that run from the desktop Windows scripting host. Our current research is focusing on inferring policies from auditing logs created from use of the Active Scripting Interface. The goal is to extract policies for safe versus malicious scripting by learning what behavior distinguishes them in logs of actual usage by malicious and benign scripts.

In conclusion, we have characterized the code-driven threat to Internet-enabled platforms as surpassing the current capabilities of anti-virus and intrusion detection tools. We also briefly described our approach to addressing one class of the code-driven threat — active-scripting-based attacks against the Win32 platform. Because our approach works at the appropriate layer of abstraction — below the application layer, but above the kernel layer — we are able to observe and control the behavior of active scripts regardless of their source, host application/interpreter, and signature. We believe this approach will be effective in addressing the next generation of code-driven attacks.

Acknowledgment

This paper describes an approach developed by members of Cigital's sandboxing team: Dur Berrier, Anup K. Ghosh, Timothy Hollebeek, and Mike Pelican.

³Symantec's list of top malware threats is available online at <http://www.symantec.com/avcenter/>