An Analysis of Mobile Code Security

Sophie Ke (26172031), Jason Kwan (39688023),
Justine Lu (41799024), Safinaaz Rawji (32307035)

Abstract – This report discusses the advantages and concerns of mobile code. In addition, three specific types of mobile code are discussed: cross site scripting, ActiveX controls, and Java applets. These were the three forms of mobile code included in our term project.

I. INTRODUCTION

The term Mobile code has continuously evolved ever since it was first introduced. From one source, it is "code that is shipped across nodes of a network, crosses protection domains, then downloaded and automatically executed upon arrival without explicit order by the recipient." In this report, it will be simply defined as applications or scripts on web pages, which are executed locally without much user knowledge or interaction[1]. If malicious mobile code travels through protective domains, such as a corporate network, all terminals connected to that node could be susceptible to being compromised. Mobile code was highly regarded as the next generation to traditional non-interactive Web applications in 2001 whereby making them rich in media, graphics, audio, and video.

Although mobile code can come in various forms, including JavaScripts, JavaApplets, ActiveX Controls, Macromedia Flash, Shockwave Files, and macros attached to Office Documents, only a select few are widely used.

As mobile code continues to gain in popularity on web pages, this report will first examine the advantages and concerns of mobile code and analyze three types of mobile code found commonly on web pages. The three forms of mobile code that will be discussed in this report are cross site scripting, ActiveX controls, and Java applets.

II. ADVANTAGES

Mobile code is flexible and supports different mediums where the computation to execute the code does not need to be known prior to execution on the recipient's system [2]. There are many advantages of mobile code including:

Efficiency: For repeated access to the same application, it is efficient to send code that can be computed at the recipient's end whereby it is executed and interact locally. This can help in networks with high latency and where interactions between server and recipient consist of few messages [2].

Simplicity and Flexibility: When code needs to be updated or a new installation is required, code is available on demand to the recipient.

Storage: Loading code on demand instead of having to install all the programs which can be duplicated on all sites. This saves total storage requirement [2].

Enhanced Web Browsing: Without mobile code, it would not be possible for popular web pages like YouTube to efficiently host media-rich content that's compatible with so many platforms.

III. CONCERNS OF MOBILE CODE

User's safety and security are compromised when mobile code contains malicious scripts that, for example, creates a covert channel (a channel that is not explicitly intended for communication [2]), allows a hacker to snoop on a user's computer, or allows a virus to propagate and affect as many different machines as fast as it can [3].

Instances where mobile code violates security policies include:

Confidentiality: When all files that are supposed to be kept confidential are leaked.

Integrity: Private data can be modified.

Availability: Denial of service attacks, whereby the hacker can stop the user in accessing certain files normally accessible to the user.

IV. CROSS SITE SCRIPTING

Cross site scripting (XSS) is one of the most dangerous and unpredictable mobile code security threats on the Internet. XSS takes place when someone gathers malicious data from other users by injecting JavaScript, VBScript, ActiveX, or HTML into seemingly safe applications, such as java applets and Flash animations, or into hyperlinks on a website. Once the user executes the application or clicks on the link, the malicious code will be able to execute locally on the user's machine, thus causing damage without the user's knowledge of its existence.

This threat is especially common as most forums and personal spaces nowadays allow users to submit responses with HTML and JavaScript embedded in them. This makes it very convenient for attackers to encode malicious scripts or links within their reply. Furthermore, attackers usually encode their malicious portion of the link in HEX or other encoding formats, thus appearing less suspicious to the users.

To hide the malicious act even more, the attacker can create an output page that appears to be valid content from the original website, or just transfers the user back to the original website after certain time limit.

Here is an example to demonstrate how you would achieve cookie theft using simple JavaScript and PHP codes [4].

First, on the website you want to steal the session cookie from, post a message with a link containing the following JavaScript:

```
javascript:document.location='/cookie.php?'+
document.cookie;
```

This will lead the user to a page named "cookie.php" with their website session cookie as the query. Once the user reaches "cookie.php" unknowingly, all of their information, including IP address, remote port, user agent, request method, remote Host, referrer, and the session cookie will be logged by a function called "logData()". Moreover, all of this information will be stored in a file called "log.txt".

A snap shot of the main functions of the code is shown below:

```
$ipLog="log.txt";
$ip = getenv("HTTP_CLIENT_IP");
$cookie =
  $_SERVER['QUERY_STRING'];
$rem_port =
  $_SERVER['REMOTE_PORT'];
$user_agent =
  $_SERVER['HTTP_USER_AGENT'];
$rqst_method =
  $_SERVER['METHOD'];
$rem_host =
  $_SERVER['REMOTE_HOST'];
$referer =
   $_SERVER['HTTP_REFERER'];
$date=date ("l dS of F Y h:i:s A");
```

Consequently, once a user clicks on the link, all of their information will be automatically logged in the "log.txt" on a remote server. The attacker can then use the information gathered to achieve account hijacking or the changing of user settings.

This threat is very difficult to prevent, and it relies heavily on the security feature and encryption power of the website, the web browser security settings, as well as your own personal discretion. You should always check the links and only follows ones you trust, or you can disable JavaScript in your web browser, which will eliminate most of the problem.

V. ACTIVEX CONTROLS

An ActiveX Control is another form of mobile code and was introduced by Microsoft in the spring of 1996 [5]. It is an object added to forms to enhance the user's browsing experience. ActiveX controls can be created in several different languages including C++, Java, and VisualBasic. Because of this flexibility, ActiveX controls can be databases and spreadsheets and they can establish connections to other computers and networks and then transfer files. In addition, all these different actions can run invisibly to the user [6]. ActiveX controls only work in Microsoft's Internet Explorer and also in Netscape, if the user has an ActiveX plug-in.

Our project contained a simple ActiveX control to demonstrate its flexibility and ability to harm one's computer if appropriate security precautions are not taken. Our control showed that a file can be downloaded into the user's computer and then accessed, and then computer settings changed. In addition, these instructions occurred invisibly to the user.

The control was created with Microsoft Visual Basic 6.0. The control downloaded a bitmap from the Internet. Specifically, the URL path of the bitmap was <http://www.ece.ubc.ca/~jmlu/412CookieMonster.bmp>. This bitmap was saved to the C:\ drive as "412CookieMonster.bmp". Then a simple instruction set this image as the computer's desktop wallpaper.

To include the ActiveX control in a webpage the author uses an OBJECT tag as shown in the following example.

```
<OBJECT ID="DemoControl"
CLASSID="CLSID:044721A2-C1AB-4CEA-8BA1-
DF2F82C9BE90"
CODEBASE="EECE412Demo.CAB#version=1,0,0,0">
</OBJECT>
```

The *ID* field is the name of the control. The *CLASSID* is a globally unique identifier used to identify the control and assigned by the authoring tool. The *CODEBASE* contains file identification information.

When a software developer creates an ActiveX control two safety settings can be set: *Safe for Scripting* and *Safe for Initialization*. Safe for Initialization means that the control will not do damage to a user's system no matter what values are passed to the control during its startup. Safe for Scripting means that the control cannot be used maliciously no matter how it is manipulated. [6] Although ActiveX controls contain these two security settings, a malicious ActiveX control developer can set these two settings even if the control isn't safe for scripting or safe for initialization.

Authors of ActiveX controls can verify their authorship by signing controls with programs such as Microsoft's Authenticode and Apple's Code Signing. Digital signatures allow a user to verify, prior to running executable code, that it came from the developer it says it came from and that nobody else has modified the code. [6] If the user trusts the signer of the ActiveX control, it will be allowed to run with full privileges [7].

There are numerous useful ActiveX controls, for example, from Imagestation.com and Xanga.com that once downloaded, enable the user to more easily upload pictures. Because of these advantageous ActiveX controls, some users may not realize the risk involved and when asked to download and run an ActiveX control from an unfamiliar webpage, may do so willingly. When an ActiveX control wants to be installed, the web browser displays a message box asking if the user wants to run the ActiveX control and if the user wants to check off the *Always trust content from* box. If the user selects this option, future controls from the same author will download and execute without warning. Therefore to secure oneself from malicious ActiveX controls, the user must properly configure their browser security settings [5]. In addition, the user must use their own judgment when accepting ActiveX controls from various websites.

VI. JAVA APPLETS

Like other forms of mobile code, Java gives developers and opportunity to integrate powerful, cross-platform programs (applets) into web pages. Aside from an enhanced web browsing experience, users also face many threats.

Some potential threats within Java, which can violate Confidentiality, Integrity, and Availability security policies [8] are:

Confidentiality: Mailing client machine information (i.e. password files,) and sending personal or company files over the network.

Integrity: Deletion or modification of files, killing processes or threads.

Availability: Creating high priority and resource intensive processes or threads, creating thousands of window pop-ups.

Even though Java was designed with security as a priority, like all other software, it also contains vulnerabilities within its design and source code. As known issues are constantly being addressed and fixed through patches and newer revisions, hackers, researchers, and other developers are constantly finding new exploits. However, assuming that there are no vulnerabilities, the Java Security mechanisms will be analyzed below.

When a web browser executes a Java applet, there are several security mechanisms in place which prevent applets from executing potentially malicious commands.

The Java Virtual Machine: The Java Virtual Machine (JVM) is intended to protect a client machine from hostile applets downloaded from un-trusted sources by only allowing the applet to operate freely within the JVM or sandbox. Some common acts which cannot be perform by default are: reading, writing, and executing files on a client system, making network connections to other computers with exception to the originating host, load libraries onto the client machine, and directly calling native methods.

Byte Code Verification: Byte code verification ensures that the Java byte code, which may not have been compiled by a Java compiler [9], conforms to Java specifications. The byte code verifier ensures that: each byte code fragment is correct, pointers are not forged, no access violations, or incorrect type object access.

The Class Loader: The class loader within the JVM is responsible for loading both user-defined and Java Application Programming Interface (API) classes into a unique namespace. The unique namespace prevents anther class with the same name from called accidentally.

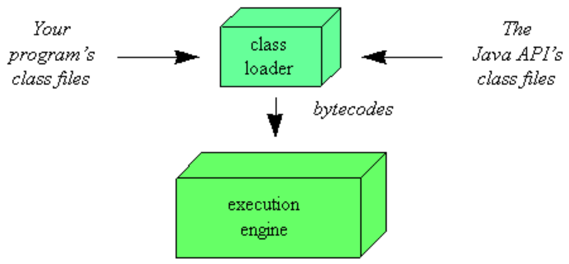


Figure 1. Java's Class Loader Architecture [10]

Security Manager: The Java security manager performs run-time checks on dangerous calls (calls which allow applets to get out of the sandbox) and generates a Security Exception. In short, it enforces applet restrictions.

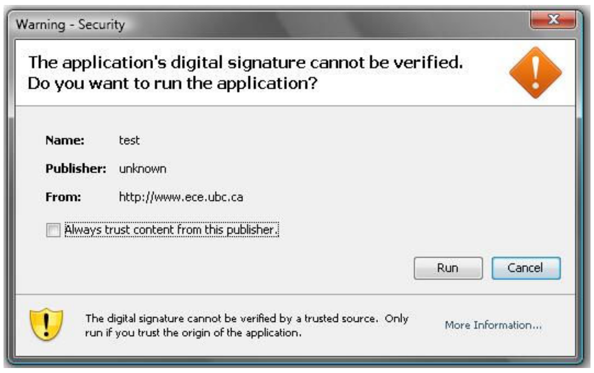
Code Signing: Code signing is a method for content publishers and software developers to digitally sign their Java applets. By obtaining a public and private key from a Certificate Authority (CA,) applets signed by this key are allowed to execute commands which unsigned applets normally cannot make. Under default browser settings trusted and signed Java applets are executed without warning.

Although code signing in Java serves as a method for legitimate applets to bypass some of the security mechanisms, it also poses the biggest problem if users allow un-trusted signed applets to run. As demonstrated in our project, anyone is capable of signing their own applet as long as they have Java Software Development Kit (SDK) installed. Simple create a key with the included keytool utility, and sign the applet with the jarsigner.

Under default browser settings, when un-trusted signed applets are loaded, a warning appears in the web browser which informs the user of an unverified applet.

If the user allows this unverified applet to run, they are essentially allowing it to run outside of the sandbox. As a result, a client machine may be compromised without even knowing.

To demonstrate this threat, a file was written to the local C drive.


Warning - Security

The application's digital signature cannot be verified.
Do you want to run the application?

Name: test
Publisher: unknown
From: http://www.ece.ubc.ca

Always trust content from this publisher.

The digital signature cannot be verified by a trusted source. Only run if you trust the origin of the application.

Figure 2. Unverified Signed Applet Warning

As Java is already quite secure, the best ways to prevent attacks from Java applets are to keep Java Virtual Machine, and web browsing software up-to-date, be aware of new security vulnerabilities, and to run signed applets with caution.

VII. CONCLUSION

As the internet continues to gain popularity, the demand for content-rich websites continues to grow. The use of mobile code is the obvious choice in meeting these demands. As discussed in this report, the advantages of integrating mobile code into web content by far, outweighs the disadvantages. In this report, three forms of mobile code, commonly found on web pages were analyzed. Cross Site Scripting attacks are probably the hardest to prevent since malicious scripts can be easily placed on legitimate servers (such as forums and blogs.) As for ActiveX controls and Java Applets, users can be easily fooled into executing these malicious controls and applets as demonstrated in our project.

In all three cases, the solutions are the same. The user must use caution when clicking on unknown links, or when running unknown or un-trusted mobile code.

REFERENCES

[1]Brown, L., “Mobile Code Security”, *School of Computer Science, Australian Defense Force Academy.* September 2004 <http://www.unsw.adfa.edu.au/~lpb/papers/mcode96.html>

[2]Thorn, T., “Programming Languages for mobile code”, *ACM Computer Surveys,* Vol. 29 No.3, 213-239. 1997

[3]Williamson, M., “Throttling Viruses: Restricting propagation to defeat malicious mobile code”, *HP Labs Briston*, August 2003

[4]Cgisecurity.com, “The Cross Site Scripting (XSS) FAQ”, August 2003 <http://www.cgisecurity.com/articles/xss-faq.shtml>

[5]Fallon, Thomas J. *The Internet Today.* Upper Saddle River, New Jersey: Prentice-Hall, Inc., 2001.

[6]Grimes, Roger A. “Chapter 11: Malicious ActiveX Controls.” Malicious Mobile Code: Virus Protection for Windows. Aug 2001. 10 Apr 2007. <http://www.oreilly.com/catalog/malmobcode/chapter/ch11.html>

[7]Rubin, Aviel D., Daniel Geer, and Marcus J. Ranum. Web Security Sourcebook. USA: John Wiley & Sons, Inc., 1997.

[8]Bank, Joseph A., “Java Security”, *MIT.* December 1995 <http://www-swiss.ai.mit.edu/~jbank/javapaper/javapaper.html>

[9]Mcgraw, G. and Felten, E., “Understanding the Keys to Java Security – The Sandbox and Authentication”, *JavaWorld.* May 1997 <http://www.javaworld.com/javaworld/jw-05-1997/jw-05-security.html?page=2>

[10]Venners, B., “Security and the Class Loader Architecture”, *JavaWorld.* September 1997 <http://www.javaworld.com/javaworld/jw-09-1997/jw-09-hood.html>