Plain English: Risks of Java Applets and Microsoft ActiveX Controls

Jennifer Marek

PLAIN ENGLISH: RISKS OF JAVA APPLETS AND MICROSOFT ACTIVEX CONTROLS

JENNIFER M. MAREK

GIAC SECURITY ESSENTIALS CERTIFICATION (VER 1.3)

4 MARCH 2002

EXECUTIVE SUMMARY

Although the technologies change, the same old issues seem to stay around. In the world of computer security, it is often the users with a new 'need' who push the network administrators for fewer restrictions.

One of the latest examples of this can be seen in the discussion of whether or not to allow 'mobile code' through the firewall onto a secure company intranet. Users need the tools to complete their work but more and more applications are being built around web browsers using mobile code in to transfer and share information over the Internet.

This paper discusses the differences between two types of mobile code, Microsoft ActiveX controls and Java Applets, and the security risks of both. Finally, the paper will gives alternative suggestions on what a can be done to allow some users to use mobile code, while not putting a secure intranet at risk.

INTRODUCTION

> *"The risks of downloading and running an unknown person's code on your machine is clear. If the code has a virus attached, it can infect your machine. If the program is a Trojan Horse, it can take over your machine for its own purposes while appearing to do something useful. How can we be sure that a program someone says is useful hasn't been hijacked to do something nasty?"* [1]

Today, organizations depend on IT systems to perform essential, mission-critical functions. So when a user wants to add a new technology onto a protected intranet, the risks must be balanced against the benefits. In fact, the new technology should only be introduced if the need is driven by a true business requirement and does not increase risk beyond an acceptable level [2].

One technology that requires a risk-benefit analysis prior to introduction onto a secure intranet is mobile code. **Mobile code is a mini-application that has a specific task**. The term 'mobile' is derived from the fact the code (mini-application) can be transferred from one computer to another with the **same or different** operating systems and/or hardware. This allows for tremendous functionality and transportability.

Imagine a program developer writing a piece an application which is **independent** of the type of computer, the operating system, or the version of other software installed on a computer. Then picture the same application being able to perform a task by itself or in conjunction with other pieces of mobile code (mini-applications) on the same computer, or with other. What this type of code can do is limitless. Unfortunately, while the functionality is increased, the risk is also increased.

This functionality has not gone unnoticed by software developers. One growing use of mobile code is in the user of application development. The use of web browsers as the interface between the user and the computer is increasing. Web browsers are like mobile code, they are independent of the type of operating system and computer hardware. This opens the available market to all computers, not just PC-based, Apple or Linux operating systems. With the web browsers acting as the interface for applications, mobile code is used to transmit and share information over the Internet for these applications. Coupled with web browsers, mobile code is a very strong tool. Information can be shared between all types of computers, running all types of operating systems.

There are several types of mobile code that can be categorized by what they can do. The Department of Defense puts mobile code into three categories [3]:

- Category 1:
  Broad functionality, allowing **unlimited access** to a user's computer and can do whatever a user can do (e.g., send e-mail, read the hard disk, surf company's intranet, change passwords, etc.). Once the code is installed on the user's machine, there is no control over what it does. There are known vulnerabilities with **few or no security countermeasures**. Examples are: (a) **ActiveX**, and (b) when used to execute mobile code: Windows Scripting Host, Unix Shell Scripts, DOS Batch Scripts.

- Category 2:
  Full functionality with **limited control** by the user on what the code is allowed to do. There are known security vulnerabilities but there are also **countermeasures or safeguards** to these. Examples are: **Java Applets**, Visual Basic for Applications, LotusScript, PerfectScript, and PostScript.

- Category 3:
  Limited functionality, which is **very restrictive** in the actions it can perform. There are known vulnerabilities, but there are **continuous security safeguards being developed.** Examples of types of Category 3 mobile code are: JavaScript, Visual Basic Script, Portable Document Format (PDF), and ShockWave/Flash.

A major risk of mobile code is that user's can download and execute the code without even being aware of what is happening since the code runs in the background. If a user doesn't know a program is running, how can a user be sure the program is not implementing destructive instructions? For example, everybody has seen the e-mails at Christmas time that are sent around which have attached files of dancing reindeer or singing Christmas trees. Embedded in these programs could be a malicious program. For example, every time the reindeer dance or the tree sings the program could scan the user's hard disk (or company secure intranet) by searching on key words, and e-mail these files to the program developer. Remember, **if malicious code can get into a secure intranet, it can get out.**

While there are three broad categories of mobile code, there are three characteristics shared by all code across category boundaries:

- The developer of the mobile code can be self-identified or anonymous – **signed or unsigned**;
- The user knows and trusts the developer or doesn't know and therefore doesn't trust the developer – **trusted or untrusted**;
- The actions of the code can be **limited or unlimited** in what it can do on the user's computer.

The two types of mobile code that will be discussed here are **Java Applets** and **Microsoft's ActiveX** controls.

ACTIVEX

The ActiveX control is a Microsoft technology that provides tools for linking desktop applications to the Internet. They can be attached to e-mail or downloaded from a known or unknown site on the Internet. What makes ActiveX control different from Applets is security. Security is the *responsibility of the user* – nothing is built into the Microsoft ActiveX development environment to protect the user from programs that are intended to do the user harm. Often these malicious programs are "hidden" in other programs to entice the user to download the code. In fact, ActiveX does not have a security model, it has a *trust* model, and the trust model is black and white: you either trust the code and let it run unhampered on your computer, or you don't. [4]

Once the ActiveX control is installed on a user's computer, it can do anything the user can do. For example, ActiveX controls can insert harmful code into the user's operating system, surf company's secure intranet, change a user's password(s), or retrieve documents off the user's hard disk or network drives and then mail them offsite.

In efforts to provide some assurance to users of ActiveX controls are safe, Microsoft has introduced an electronic process to identify the developer of the code and provide some assurance the code has not been changed. This is done through digital signatures and digital certificates. However, digital signatures and certificates are only **precautions**, not guarantees the code is safe.

For example, last year a Microsoft digital certificate was "stolen" by a computer hacker and used to gain Microsoft customers trust.[5, 6] This is proof a signature can be spoofed or stolen. This example reinforces the phrased used in the ActiveX security workshop report: "The ultimate problem with any signature scheme is that safe controls can come from untrusted sources, and unsafe controls can come from trusted sources." [7]

At the time of this writing, MITRE Corporation has published over 31 common vulnerabilities and exposures (CVE) listed for ActiveX controls. [8] **MITRE** Corporation is a not-for-profit organization that provides systems engineering, research and development, and information technology support to the government. It operates federally funded

research centers for the Department of Energy, the Federal Aviation Administration and the Internal Revenue Service.

In using the general categories of high, medium and low, the National Institute of Standards and Technology (NIST) has assigned a risk level for ActiveX controls as a **HIGH**. Although a relative term, here "high" refers to a **threat-source** that is highly motivated and capable. Controls to prevent stop the threat-source are not effective. Once the attack is launched, it may result in the (1) loss of major tangible assets or resources; (2) harm or impede and organization's mission, reputation, or interest; or (3) in human death or serious injury.[9]

A comprehensive list of ActiveX concerns is in Appendix A.

JAVA APPLETS

Java Applets are also mini-applications. They are developed using Java and have a structured security environment in which the developer can implement specific security rules for the Applets to follow once they are downloaded to a user's computer.

Additionally, Java Applets should not be confused with JavaScript. These are two different things, developed by different companies. JavaScript is a product of Netscape.

If the user does not 'trust' the an Applet, Java security model requires the untrusted Applets to be only run in a "sandbox" where its actions are limited. This puts a lot on the user. Many users will click *yes* to any message/checkbox so they are not hindered in viewing a web page.

A **sandbox** is a 'virtual' boundary for a computer application (or mini-application like an Applet). The sandbox was designed and is programmed into Applets to give the user some reassurance the Applet won't do harm to a user's computer. The Applets in a sandbox are limited in what it can do – e.g., what the Applet can read and where it can write.

However, hostile Applets still pose a security threat, even within a sandbox. An Applet can consume or exploit resources improperly or cause a user to perform actions he might not normally do. Some examples of hostile Applets exploits include: denial of service attacks, mail forging, etc. Also, many bugs have been found in the implementation of Applets that will allow the security features to be sidestepped. [10]

Java security has been broken many times because of developers not implementing their code based on the Java security model.[11] At the time of this writing, MITRE Corporation has published over 14 CVEs for Java Applets.[12]

In using the general categories of high, medium and low, NIST has assigned risk level for Applet controls as **MEDIUM**. The definition of "medium" is a motivated and capable threat-source, but controls are in place that may stop an attack. If the attack was successful, the attack may result in: (1) the loss of expensive of assets or resources; (2) the violation or harm of an organization's mission, reputation, or interest; or (3) human injury. [13]

DIFFERENCES BETWEEN JAVA APPLETS AND ACTIVEX CONTROLS

Java Applets and ActiveX controls are similar in the sense they are a type of mobile code. They both are mini-applications developed to perform a specific task. However, the similarity stops there. The security for the Java Applets and ActiveX controls is significantly different. Security for ActiveX controls is all or nothing. By loading an ActiveX control on to their computer, the user has made the decision to trust the control not to do anything bad. Once loaded, there is no boundary to what the control can do. ActiveX controls are downloaded one, and remain on the computer until removed by the user.

Contrary, Applets have security built into their design. The user can often define what the Applet can and cannot do. Additionally, Applets are downloaded into the user's computer RAM. Therefore, once the computer is shutdown, or restarted, the Applet goes away. However, only the Applet is gone, the actions taken by the Applet while it is in RAM are not undone.

A limited analogy, which might help to explain the difference in security controls between the two types of mobile code, is a traveler on an airplane. A traveler buys a ticket to go to Japan.

After the traveler is allowed into the airport, this is the result if the traveler uses Microsoft's *ActiveX control rules*:

- There are no ticket collectors to see if he is getting on the correct airplane
- No airport security
- The traveler can go where he wants, on any airplane, to any destination, and do anything he wants – there is nobody to stop him
- The traveler can even fly the airplane if he wants to
- The traveler is **trusted** to behave.

Similarly, once an ActiveX control is granted access to the user's computer, there is no way to force the control to obey any rules and do only what the user thought it would do. The ActiveX control is **trusted** to behave.

If the traveler is using the *Java Applet's rules*,

- Ticket collectors check to see if his ticket allows him access to the airplane he is trying to get on and only allows him to board the airplane his ticket allows him to board;
- Airport security is there to make sure the traveler obeys the rules while in the airport;
- The traveler is limited to the passenger areas of the airport and airplane and this is enforced by airport security;
- The traveler must obey the specified rules as defined by the airport.

Similarly, Applets must obey the security rules that are put into place by the developer, using the Java security model. This security model not infallible - implementing the security model is complicated and rules are not always followed. Additionally, as with airport security, **Java security can be sidestepped** and there is always the danger the user's computer, or company's network, can be hijacked.

Although this analogy is limited and cannot go fully into all the nuances of ActiveX and Applets, it does show the basic differences in the security of each.

JAVASCRIPT

For the sake of completeness, a comparison between Java Applets and JavaScript is being included. As stated earlier, **Java Applets** are 'mini-applications' which are developed using Java, a Sun Microsystems technology. Applets are separate pieces of software that need to be downloaded to be used. **JavaScript** is a Netscape product and is totally different from Applets. Unlike Applets, JavaScript does not create mini-applications. Instead, JavaScript is code that is *embedded* into web pages. When web pages are downloaded, the code that formats the page for viewing, Hypertext Markup Language (HTML), can include JavaScript. The actions are very limited and most importantly, JavaScript does not have the capability to allow remote users onto a computer.

Some examples of what JavaScript can do are: pop-up messages when a user's mouse rolls over a picture on a web page, display the current date on a web page, or display a scrolling message at the bottom of the web page.

JavaScript has limited functionality and cannot allow a person from outside the intranet access to the intranet without permission. For this reason the DoD feels JavaScript only poses a limited risk to the user and the user's network.[14]

MANAGING THE RISKS

In order for an intranet to be secure and the risks that do exist to be managed and reduced to an acceptable level, the type of attacks that a mobile code can initiate must be understood. There are four basic types of attacks: (1) attacks which modify a system (user's computer, network server, etc.); (2) attacks which invade the user's/company's privacy; (3) attacks that use up all the available resources to as not to allow anyone else to user the system; and (4) attacks which are annoying but don't cause any real harm.[15]

The first type of attack, modification to a system, is the most dangerous. Memory can be erased, processes can be killed, data can be altered, and once one piece of data is called into question, all data is suspect. Applets and ActiveX are capable of launching this type of attack. When correctly designed and implemented, a sandbox can limit Applets actions. However, ActiveX has no safeguards against this type of attack.

Invasion of the users privacy is almost as dangerous as system attacks. Company private information can be sent via e-mail to the general public. Financial data could be stolen from a competitor and used to the advantage of the company who initiated the attack through mobile code. Businesses would loose customers if the customers did not feel their credit card numbers were not safe on the business' computers. Although neither the systems nor its data are damaged, this type of attack can seriously damage a company is sensitive data is made public.

Denial of service (DoS) attacks are annoying and can tie up a company's computer resources fighting the malicious code, instead of servicing customers. The computers and the networks don't actually come to harm. DoS attacks are not as serious as the first two attacks described but they can cost a company customers and revenue.

The final type of attack which mobile code can launch is annoying attacks that don't cause any harm, but interrupt the business day. An example of this type of attack is the button that jumps around the screen that the user can't click to shut down a program. Or unwanted sounds or pictures could popup on a user's screen. Worker productivity could decline if every five minutes the screen goes blank or an unwanted picture pops up. The cost in network administrator time alone is enough to cost the company money.

The security risks outlined above are not new. What is new is the way in which these types of attacks can be launched. To launch an attack on a computer system, the attacker must be inside the system. If the attacker can't get inside, an agent (ActiveX control or Applet) of the attacker can do just as much harm, and so much the better if the agent is invited into the system.

So why do people insist on using mobile code even knowing the security risks? The part of the answer is mobile code is highly functional and reusable. Once an Applet or ActiveX control is written, it can be reused over and over again. Also, for novice programmers, 'slick' features can be added quite easily using code written by someone else. Additionally, ActiveX controls can be shared among Microsoft applications (Access, Excel, Word, etc.).

Also, as stated in the introduction of this paper, mobile code is being used with new software applications that are web browser based. This increases the functionality and the efficiency for a application which must share data over the Internet.

However, user be warned – the tool of choice for hackers is mobile code. In fact, so many of the hacker attacks use mobile code to the extent the National Security Telecommunications and Information Systems Security Committee (NSTISSC) recommends using older browsers, such as Mosaic, for use in DoD [16]. [Note: Under Executive Order (E.O.) 13231 of October 16, 2001, "Critical Infrastructure Protection in the Information Age", the President has redesignated the **National Security Telecommunications and Information Systems Security Committee** (NSTISSC) as the Committee on National Security Systems (CNSS).[17]]

The older browsers do not support mobile code. While this may appear to be an extraordinary measure, put into the context of network sensitivity that contains trade secrets, pricing information, etc., it indicates the seriousness of the risk incurred using mobile code. Many organizations have chosen to mitigate the risk of mobile code by using the latest version of web browsers, but filter out all Java Applets and ActiveX controls to the extent possible.

Security involves constantly evaluating and mitigating risks. The risk of allowing Java Applets and ActiveX controls through a firewall indiscriminately is **HIGH.** Accordingly, the overall need to introduce mobile code an intranet should be sufficiently justified through a **business case,** a **risk assessment** and a **risk mitigation plan.**

The risks can be reduced if precautions are taken, but many times the precautions are difficult to enforce. As stated earlier, the DoD categories mobile code into three categories. Also stated earlier are three characteristics which mobile code can have. If mobile code's risk is assigned using these categories and characteristics, as the DoD has done, the risk of allow mobile code through a firewall may be reduced.

The DoD has chosen reduced the risks by implementing the following policy:

DoD Mobile Code Policy [18]				
Category 1 allowed if:	Signed	Trusted source	Approved PKI signing certificate	Unsigned requires waiver
Category 2 allowed if:	Signed	Trusted source	Over an "assured channel"	
	Unsigned	Stand-alone computers only		
Category 3	Maybe used			
Mobile code in e-mail	'…automatic execution of all categories of mobile code in e-mail bodies and attachments shall be disabled."			

The definition of an assured channel referenced in Category 2 above is a *network communication link* that is protected by a security protocol providing authentication and data integrity, and employs US Government approved cryptographic technologies whenever cryptography is used.

The following protocols and mechanisms meet the requirements of authentication and data integrity protection for an assured channel:

- Internet Protocol Security (IPSec)
- Secure Sockets Layer (SSL)
- Transport Layer Security (TLS)
- Secure Multipurpose Internet Mail Extension (S/MIME)
- Digital code signing using DoD-approved PKI signing certificate or other National Security Administration (NSA)-approved high assurance guards with link encryption methodology.

These extreme measures show how seriously the DoD takes the threat risk of using mobile code. Unfortunately, not all companies can implement such an expensive solution and resort to using firewalls which can be configured to allow only 'signed and trusted source' pieces of mobile code onto the intranet, or not. Often it is an all or nothing type of configuration. All types of ActiveX controls may pass through the firewall, or no ActiveX controls may pass through the firewall.

ALTERNATIVES

While the risks for allowing mobile code into an intranet are high, there are some alternatives in providing this service to users.

1. Employ the use of modems:
   A stand alone computer connected to a modem would allow a user to connect directly to the network which requires the use of mobile code. This option would not put the company's intranet at risk by isolating the user from the intranet, while allowing the user access to the Internet sites that use mobile code.

2. Separate network outside the intranet:
   Establish a parallel network for a limited number of users. This "green" network could be connected to a company's Internet connection at their DMZ. This would provide firewall protection for the green network. While this option is costly, it would not put the intranet at risk.

3. Establish one or several terminals with a direct connection to the Internet. If there are no disk drives in which staff can down load information to the transfer it to their desktop, theoretically the secure network is safe.

FINAL WORD

Be aware! If a network doesn't meet the needs of the user, the user will find a way to have his/her needs met. If mobile code is blocked for an intranet, and the user strongly feels the information is needed, he/she will download this information at home, and just bring it in on a disk. If a malicious code is present on the disk, it will then be present on your secure intranet and all the safeguards that the network security staff has put into place will be for nothing.

APPENDIX A

A COMPREHENSIVE LISTING OF ACTIVEX SECURITY CONCERNS[19]

1. ActiveX controls run directly on the users hardware and can be invoked remotely (via the Internet).

2. Microsoft uses the digital signature technology Authenticode. However a digital signature or certificate does not guarantee the control is safe. It is up to the user to decide if the control is safe.

3. Avoiding Internet Explorer and Outlook does not make a user safe from ActiveX problems. Many third party applications use the ActiveX controls as part of their ordinary operation.

4. The decision to install software should be based on the capability of the software. The capability cannot be assured with ActiveX controls. Instead, the decision to install the control must be based on the source of the control, which has been proven not to be trustworthy.

5. An ActiveX control has all the privileges of the current user and there is no mechanism for restricting the privileges of the control.

6. There is no security model for ActiveX; ActiveX has a trusted model. Either you trust the control or you don't.

7. Scripts can use controls in ways unanticipated by the original control author. This leads to unexpected behavior that can be used to violate security policy.

8. Scripts can invoke controls to run in the background. A user might not even know a script is using controls, and it may be impossible to determine beforehand which controls are being used (as in an HTML document). As a result, a user might not be able to make informed decisions on whether to open such documents.

[1] McGraw, Gary and Edward Felten, "Mobile Code and Security", IEEE Internet Computing, Nov./Dec. 1998. URL: http://www.computer.org/internet/v2n6/w6gei.htm

[2] National Security Telecommunications and Information Systems Security Committee. "Advisory Memorandum on Web Browser Security Vulnerabilities." NSTISSAM INFOREC3-00, August 2000. URL: http://www.nstissc.gov/Assets/pdf/NSTISSAM_INFOSEC_3-00.pdf

[3] Money, Arthur, Assistant Secretary of Defense. "Policy Guidance for use of Mobile Code Technologies in Department of Defense (DoD) Information Systems." 7 November, 2000, 50 http://www.c3i.osd.mil/org/cio/doc/mobile-code11-7-00.html

[4] McGraw, Gary and Edward Felten, "Securing and Managing Mobile Code." 1998. URL: http://www.cigital.com/javasecurity/compstrat.html

[5] VeriSign. "VeriSign Security Alert Fraud Detected in Authenticode Code Signing Certificates." 22 March 2001. URL: http://www.verisign.com/developer/notice/authenticode

[6] Symantec. "Fraudulent Digital Certificate." 23 March 2001.
URL: http://www.symantec.com/avcenter/sirc/fraudulent.digital.certificate.html

[7] CERT Coordination Center, Carnegie Mello University. "Results of the Security in ActiveX Workshop." December 2001. URL: http://www.cert.org/archive/pdf/activeX_report.pdf

[8] Common Vulnerabilities and Exposures (CVE). CVE version 20010918.
URL: http://cve.mitre.org/cgi-bin/cvekey.cgi?keyword=java+%26applets

[9] Jansen, Wayne A. "Guidelines on Active Content and Mobile Code." National Institute of Standards and Technology (NIST). Special Publication 800-28, October 2001.
URL: http://csrc.nist.gov/publications/nistpubs/800-28/sp800-28.pdf

[10] Ibid.

[11] Scambray, Joel and Stuart McClure, George Kurtz. Hacking Exposed: Network Security Secrets & Solutions. Reading: McGrawhill. 2001. 614

[12] CVE version 20010918.

[13] Jansen.

[14] Money.

[15] Byte. "How to minimize the risks of executable content." May 1997.
URL: http://www.byte.com/art/9705/sec7/art9.htm

[16] National Security Telecommunications and Information Systems Security Committee.

[17] Executive Order 13231. "Critical Infrastructure Protection in the Information Age." 16 October 2001. URL: http://www.whitehouse.gov/news/releases/2001/10/print/20011016-12.html

[18] Money.

[19] CERT Coordination Center, Carnegie Mello University.

REFERENCES

1. McGraw, Gary and Edward Felten, "Mobile Code and Security", IEEE Internet Computing, Nov./Dec. 1998. URL: http://www.computer.org/internet/v2n6/w6gei.htm

2. National Security Telecommunications and Information Systems Security Committee. "Advisory Memorandum on Web Browser Security Vulnerabilities." NSTISSAM INFOREC3-00, August 2000. URL: http://www.nstissc.gov/Assets/pdf/NSTISSAM_INFOSEC_3-00.pdf

3. Money, Arthur, Assistant Secretary of Defense. "Policy Guidance for use of Mobile Code Technologies in Department of Defense (DoD) Information Systems." 7 November, 2000, http://www.c3i.osd.mil/org/cio/doc/mobile-code11-7-00.html

4. McGraw, Gary and Edward Felten, "Securing and Managing Mobile Code." 1998. URL: http://www.cigital.com/javasecurity/compstrat.html

5. VeriSign. "VeriSign Security Alert Fraud Detected in Authenticode Code Signing Certificates." 22 March 2001. URL: http://www.verisign.com/developer/notice/authenticode

6. Symantec. "Fraudulent Digital Certificate." 23 March 2001. URL: http://www.symantec.com/avcenter/sirc/fraudulent.digital.certificate.html

7. CERT Coordination Center, Carnegie Mello University. "Results of the Security in ActiveX Workshop." December 2001. URL: http://www.cert.org/archive/pdf/activeX_report.pdf

8. Common Vulnerabilities and Exposures (CVE). CVE version 20010918. URL: http://cve.mitre.org/cig-bin/cvekey.cig?keyword=%22java+applets%22

9. Jansen, Wayne A. "Guidelines on Active Content and Mobile Code." National Institute of Standards and Technology (NIST). Special Publication 800-28, October 2001. URL: http://csrc.nist.gov/publications/nistpubs/800-28/sp800-28.pdf

10. Scambray, Joel and Stuart McClure, George Kurtz. Hacking Exposed: Network Security Secrets & Solutions. Reading: McGrawhill. 2001. 614.

11. Byte.com. "How to minimize the risks of executable content." May 1997. URL: http://www.byte.com/art.9705/sec7/art9.htm.

12. Executive Order 13231. "Critical Infrastructure Protection in the Information Age." 16 October 2001. URL: http://www.whitehouse.gov/news/releases/2001/10/print/20011016-12.html