Mobile Code Threats, Fact or Fiction

Carey Nachenberg
Symantec AntiVirus Research Center
Cnachenberg@symantec.com

Presented at the International Virus Prevention Prevention Conference (IVPC), 1999

And posted with the permission of ICSA

Table Of Contents

Introduction

"Don't run with scissors."
"Don't tease the animals."
"Chew your food."

As children, most of us heard advice like this more times than we care to remember. And while it's good to be safe, let's be honest, kids will do what kids will do. There are obvious risks associated with each of the above actions, yet most children grow up to be healthy adults, even after repeatedly violating our parents' helpful hints. Ignore the warnings above and you're subject to risk: tease the dog and he will bite, swallow your food whole and you'll choke; yet most of us have done these things and survived to tell about it.

Browse the web and you'll get hit by malicious Java. Surf an unknown web-site and an ActiveX component will send your spreadsheets to Bulgaria. Some of the mobile code technologies emerging now are chock-full of security holes – and pose as much risk to our computers as scissors do to our bodies. Some of the technologies are better than others, but all have vulnerabilities that need to be understood if we are to deploy them and want to maintain our company's security. The interesting thing is, while the potential for danger is great, the number of recorded incidents of people affected by malicious mobile code is extremely small, in fact *ZERO* to date[i]!

While we have seen hundreds of "proof of concept" malicious mobile applets, one is hard-pressed to find even one "wild" malicious mobile code incident! At the Symantec AntiVirus Research Center (SARC) we have never received a malicious Java or ActiveX program from any customer, corporate or end-user. In fact, after two days of research, I could only find one fuzzy reference to a possible real-world Java or ActiveX-based attack.

Where does this leave us? Given the dearth of confirmed, real-world attacks, this paper discusses the future potential for attacks by each of the popular mobile code systems. Following this discussion of threats, the paper will cover the available anti-malware technologies which can be deployed in the enterprise to address these invaders. Finally, we will provide a list of increasingly obtrusive policy suggestions on how to safeguard your organization from malicious mobile code. The solutions you choose will based on the security requirements of your organization and on how obtrusive you are willing for your solution to be to users and business partners.

Stationary Threats vs. Anonymous Threats

Why have we seen no "wild" attacks using mobile code such as Java, JavaScript or ActiveX when there have been tens of thousands of confirmed "wild" attacks by computer viruses, worms (such as HAPPY99.EXE) and Trojan horses (e.g. PICTURE.EXE). We believe that the main reason for this lack of real world mobile code attacks has to deal with a sociological effect rather than a technological cause. To understand this effect, let us classify mobile code threats into two sociological categories: stationary mobile code and anonymous mobile code.

Malicious Java applets, ActiveX, JavaScript, Jscript and VBScript are all examples of *stationary threats*. This terms means that the mobile code is retrieved from a fixed or stationary location - in this case a web-site - which can be tracked and has liability for any malicious content posted on the site. Should an end-user surf to a site and be targeted by malicious ActiveX, they could easily locate that site again and use litigation to obtain compensation for any damage done by the malware. Unlike viruses, these types of

malicious mobile code don't spread on their own and therefore need to be posted with intention on a stationary site.

In the future, we can expect the number of stationary threats to grow [ii], but still at a much slower rate than the observed growth-rate of viruses and *anonymous threats*. The likelihood is that the biggest growth of stationary threats will be due to professional scams. For example, a con artist might set up a web site with a similar name to an existing banking web-site: www.citybank.com instead of www.citibank.com. If a user inadvertently surfs to this site, the scam artist can provide fake login screens and obtain the user's banking information. This can then be used to steal money from the user's account and shortly after, the scam-artist and his web-site will disappear. These professional scams will grow in number as more malevolent people recognize the spoils that await them on the Internet.

How about anonymous threats? Anonymous threats are those threats which are delivered in such a fashion that the receiver of the mobile code cannot identify the origin of the threat or assign liability to any one person. Computer viruses and worms fall under this category, since they can spread on their own and it is difficult to track their origin. Furthermore, Trojan horse programs which are delivered in anonymous e-mail or on the Internet USENET news groups also fall into this category. Using these avenues of distribution, it is virtually impossible to determine where these threats came from making malicious individuals are more inclined to distribute their malware. We expect the number of anonymous threats to grow, and the number of users actually hit by these threats to grow as well (most of the users just logging onto the Internet for the first time are novices and have not been trained in safe computing practices.)

In addition to the obvious anonymous threats, such as Trojan horses sent in anonymous e-mail, there are a number of anonymous threats which can be distributed in such a way that they appear to be stationary; this may give the end-user or corporate user a false sense of security. For example, a malicious individual can obtain a personal web-site from companies like GeoCities. The individual has to provide little, if any, identifying information to GeoCities to obtain his or her own web site. The individual can then distribute a malicious ActiveX or Java applet on their web page and have virtually no risk of being caught. If someone realizes their web-site harbors malicious code, they can *disappear* and start a new web site with another host.

The HAPPY99.EXE worm is another example of an anonymous threat which guises itself as a stationary program. When executed on an uninfected system, this worm will install itself into the user's operating system. The next time the user sends e-mail, the worm appends itself to the e-mail as an attachment. When the recipient receives the e-mail, they assume the attachment is a legitimate program from their friend and feel little or no risk in running the HAPPY99 program. Ironically, this feeling of security - "If my friend tried it and sent it to me, it must be OK" – makes people more inclined to run the program and spread the threat.

Since Java, ActiveX, JavaScript, JScript and VBScript are inherently delivered from stationary sources (i.e. web-sites), the liability aspect will definitely limit the growth and severity of these threats. As we will see, a number of technologies such as digital certificates and code signing have been built to ensure that the web-going public can identify the source of web-based content. While these technologies cannot prevent attacks in a strictly technological way, they definitely reduce the likelihood of mainstream attacks based on largely sociological grounds.

Payload Classes

Malicious mobile code can be categorized into three different classes based on its *payload*. The term payload is used to describe the malicious behaviors that the mobile code performs which is not authorized by the user. [iii]

System Modifications

Any mobile code that modifies the client's computing environment can be said to perform system modifications. These system modifications can be as simple as deleting a file or as complex as installing a virus or worm into the client operating system. The following areas of the computer can be modified or destroyed by mobile code:

1. The malware can alter the system registry.
2. The malware can specify which files to launch during start-up.
3. The malware can modify batch files and .SYS files, such as AUTOEXEC.BAT or CONFIG.SYS.
4. The malware can delete, format or modify the contents of the hard drive.
5. The malware can re-partition the hard drive.
6. The malware can update or subvert the machine's web browser security settings.
7. The malware can introduce additional ActiveX and Java components to the system to produce back doors for future attacks.
8. The malware can subvert anti-virus software which is not aware of the new mobile code threat.
9. The malware can modify or delete the contents of other file servers or peer to peer servers depending on their security settings.
10. The malware can be re-write flash BIOS chips.
11. The malware can introduce a virus, worm or logic bomb by modifying existing system files or adding new system files.
12. The malware can modify data files to cause an application to behave in an unanticipated way. For instance, the Chaos Computer Club posted a demonstration ActiveX program which would update Quicken's data files to cause Quicken to transfer funds during the next on-line transaction session.
13. The malware can modify the public and private key information found on the machine.

As we will see, different mobile code systems have different security infrastructures which either hamper or facilitate the modification of the client computer's resources. Not all classes of mobile code can perform all of the above attacks.

Invasion of Privacy

In addition to destroying or modifying data, mobile code can export information from an affected system to any number of destinations on the Internet. Malware such as PICTURE.EXE have demonstrated how easy it is to launch this type of attack [iv]. In general, the following types of information can be exported by mobile code:

1. Documents, spreadsheets, databases and other data files – such as Quicken files and graphics files - can be exported.
2. Windows passwords and other passwords can be found on the hard drive or obtained via "key grabbing" malware. [v]
3. Conversations can be recorded via now-standard PC microphones.
4. The malware can export a list of the software (and their version numbers) being used on a machine for advertising and demographics purposes.
5. The malware can export the public and private key information found on the machine.
6. The malware can determine which web sites you have surfed to and what information has been sent to these sites.
7. The malware can forge an e-mail from your e-mail account and send it to any number of people.

Again, different mobile code systems have different security infrastructures which will affect malicious software's ability to export information from a targeted computer. Not all classes of mobile code can perform each of the above attacks.

Denial of Service

Denial of service describes a set of attacks which prevent a client computer from performing its usual duties, due to the actions of malicious mobile code. Most of the malicious Java applets we have seen have implemented denial of service attacks. These applets will attempt to allocate all available memory or create thousands of windows on the client machine to prevent other programs from running properly. Other malicious code may attempt to close the Internet connection if the user surfs to a specific site or tries to read his or her e-mail.

While denial of service attacks can be carried out by virtually all of the mobile code types, most denial of service attacks have been implemented in Java. Java, as we will see below, is a fairly secure mobile code platform and many of the more malicious attacks (such as invasion of privacy or system modification attacks) are impossible to implement. Consequently, malware authors have implemented the easier-to-program denial of service attacks in Java.

Threat by Platforms

Java

Let's examine how Java applets are used by the client computer in a typical web browsing situation:

1. The web browser connects to a web-site server, for instance www.javathreat.com.
2. The web browser downloads the appropriate web page, for instance *main.html*; the web page contains a reference to a Java file or a Java archive (.JAR) file which can contain multiple pieces of a Java applet as well as digital certificate information.
3. The web browser downloads the appropriate Java file or JAR file.
4. If the applet is provided in a JAR file, the web browser checks to see if a digital certificate is included.
5. Based on the security settings of the browser, the validity of the digital certificate and possibly the client user's input, the Java applet is either accepted or rejected.
6. The browser then inspects the "byte code" contents of the applet to verify that it adheres to Java standards and doesn't try anything malicious.
7. If the applet meets security requirements, the web browser loads the Java applet into the Java virtual machine (sandbox) and runs it.
8. The Java code is not permanently maintained and must be re-downloaded during subsequent surfing sessions to be used.

Now that we have an understanding of how Java content is used during web surfing, let's examine its potential as a carrier malicious mobile code.

Java has received the short end of the stick when it comes to its perceived risk in the enterprise. It's unfortunate because Java *running in a properly configured web browser* is by far the safest type of mobile code available today and was designed from the ground up to provide robust security.

With all this security, why does Java receive such a bad rap? As the first mainstream mobile code system which attempted to address security issues, it has come under a great deal of scrutiny. From February of 1996 to August of 1997, thirteen different security holes were found in various implementations of Java.[vi] The vast majority of these security holes were due to improper implementation

rather than a fundamental design flaw in the Java language/system. This fact surely comforts the designers of the Java system but may not be consolation for system administrators.

The Java security model was designed to protect users against the two most dangerous classes of attacks described above: namely invasion of privacy and system modifications. While Java provides robust protection against these attacks, it provides little or no protection against denial of service attacks. This is why most of the malicious Java applets posted to the web implement simple denial of service attacks. Unfortunately, the thirteen security holes mentioned above addressed more serious flaws in Java implementations (such as Navigator and IE) which allowed invasion of privacy and system modification to take place.

The question arises: Is Java safe enough for my enterprise? Unfortunately, it's difficult to give a yes or no answer to this question. Researchers continue to find holes in Java security, albeit at a far slower rate[vii], which points to an increasing level of security and robustness of the Java platform and its implementations. **Given the slowdown in discovery of new security holes, it is our belief that Java running under a properly configured web browser provides a very secure environment for mobile code with respect to both invasion of privacy and system modification attacks.** If this is your primary concern, allowing Java into your enterprise is a safe bet. However, if you are concerned about your users encountering denial of service applets, you should think twice about implementing Java. Java has been and continues to be susceptible to a wide variety of these attacks. Luckily, terminating the web browser or rebooting the system can easily deal with any of the denial of service attacks.

Signed vs. Unsigned Java

Signed Java describes Java applets which are accompanied by a digital signature or certificate information, allowing the client to determine the origination and authorship of the applet. The digital signature allows the web browser to certify that the Java applet's binary contents match 100% with the applet originally signed by the software provider. Depending on the security settings in the browser, the administrator (or end-user) can allow such signed applets to have extensive access to the local computer and the network. For more information on code signing and digital certificates, see the *Digital Signatures* section below. In general, the administrator or end-user can permit signed Java applets to do any of the following to a host system[viii]:

1. Read files on the client file system.
2. Write files to the client file system.
3. Delete files on the client file system.
4. Rename files on the client file system.
5. Create a directory on the client file system.
6. List the contents of a directory.
7. Check to see whether a file exists.
8. Obtain information about a file, including size, type, and modification timestamp.
9. Create a network connection to any computer other than the host from which it originated.
10. Listen for or accept network connections on the client system.
11. Create a top-level window without an untrusted window banner. (All windows created by an applet are labeled as insecure so the user knows where they came from.)
12. Obtain the user's username or home directory name through any means.
13. Define any system properties.
14. Run any program on the client system.
15. Make the Java interpreter exit.
16. Load dynamic libraries (DLLs) on the client system.
17. Create or manipulate any thread that is not part of the same applet.
18. Install permanent new Java components on the system.

Unlike signed applets, unsigned Java applets are not accompanied by signature or certificate information. Consequently, these applets are treated with the highest level of security when running in the web browser, since their origin and author cannot be verified. Unsigned Java applets are restricted from accessing any of the resources listed above and are terminated immediately if they attempt any access. There is one exception to this, as you will read below.

Java Back Doors

For Java applets to properly run, they must rely upon a number of built-in Java modules which are typically installed during installation of the web browser. These built in modules provide many of the basic functions that Java applets need to work properly, such as popping up windows, displaying graphics, opening files, opening network connections, etc. These modules integrate with the web browser and are specifically designed to be secure and not open up any holes to the client computer. For instance, any time an applet tries to use a built-in module to open a file, the module will check the security level of the applet and deny the request if its security is not appropriate (in the case of an unsigned applet, the request to open a file would always be denied).

While the modules that ship with common web browsers are considered very safe, it is possible (and actually very easy) for Java developers to install their own built-in modules and compromise security. Standard browser implementations of Java will allow Java applets to use any and all installed Java modules as long as they are in one of a specified set of directories called the *CLASSPATH*. Specifically, developers can add their own home-brewed Java modules to a computer and then set an environment variable, called CLASSPATH, to point to the directories where these modules are stored. Here's an example of how one might set a CLASSPATH variable in the AUTOEXEC.BAT:

```
SET CLASSPATH=C:\MyClasses; C:\OtherClasses
```

This CLASSPATH tells Java development environments *and Java browsers* that they can find additional Java modules in the C:\MyClasses directory, as well as in the C:\OtherClasses directory.

Common web browsers allow a Java applet to use any Java modules that are in directories specified by the CLASSPATH as if they were tested, secured modules. Unfortunately, Java developers can install (to these directories) any number of modules that have not been scrutinized and may open up numerous security holes. For instance, a Java developer may create a Java module that can update the registry. If the developer placed this module in a directory specified in the CLASSPATH, so his other Java programs can use it, he would also implicitly grant access to any Java applet from the web that wanted to use the functionality as well! If an attacker knew that the user had this module on his machine, the attacker could design an applet which will call upon this module to update the client computer's registry!

There are several possible ways to defend against such a CLASSPATH attack:

1. Certify the security of all Java modules in the CLASSPATH of a given computer.
2. Disallow computers with a CLASSPATH variable (i.e. those that have Java development environments or other Java applications) from using Java in the web browser.
3. Make sure to erase the CLASSPATH variable before web browsing, and reset it after browsing.

This hole constitutes a real potential threat to web surfing users, although the likelihood that it will be exploited is very small. This is one of the only serious Java holes which has not been *fixed*.

Java and Malware

As we have seen, unsigned Java is a safe medium and is designed to make the two worst types of attacks - system modification and invasion of privacy - virtually impossible. For this reason, the biggest

threat from Java has been, and continues to be, denial of service attacks. The author has been unable to locate denial of service attacks outside of research web sites and believes that this is a very low risk.

Unlike unsigned Java, signed Java provides a much larger risk to corporations and end-users. Like ActiveX, signed Java can access many sensitive areas of the computer without restriction and can implement system modification and invasion of privacy attacks. **Therefore, from a strictly technical point of view, signed Java does pose a risk to corporations and end-users.**

Luckily, Java malware is by definition a *stationary threat*, and as such, ensures a certain amount of liability for the web site operator who posts it. We believe that this liability will help to curb the growth of Java malware going forward. But it is by no means a guarantee of safe computing.

Java Viruses

A Java virus falls under the category of a system modification attack since the virus must modify the client system in order to infect it. A Java virus may also fall under the other malware categories, depending on its payload. Since Java is virtually immune to system modification attacks, Java viruses, as we will see, are not a viable threat to the end-user or enterprise.

To date, we have seen three viruses written for the Java platform: Strange Brew, JavaVirus and BeanHive. Fortunately, the first two of these Java viruses (Strange Brew and JavaVirus) are completely neutered by the standard security provided by the popular web browsers. These viruses attempt to open files, which is strictly forbidden by the Java implementations of the major web browsers, and are immediately terminated.

The third virus, BeanHive, is actually delivered as a *signed* Java applet. Signed Java applets can explicitly request access to the client computer's files and other resources. The BeanHive virus does just that; it requests permission to modify files on the system as soon as it is downloaded to the user's computer. When the virus makes this request, the user is provided with a dialog box and can choose to grant or disallow the access. Therefore, for the BeanHive virus to spread, the web-browsing user must explicitly grant access to the digital certificate that accompanies the infected Java files. If the user does grant access, the virus is able to open and infect files on the client computer. However, due to bugs in the virus, these infections are unable to properly spread on their own and constitute no threat to the user. If the user does not explicitly grant access, the virus is immediately terminated in the web browser.

Even if a virus like BeanHive did infect other Java applets properly, it would run into additional problems. If such a Java virus were to locate another signed applet and modify it, then the just-infected-applet's binary contents would no longer match the contents of the digital signature [ix]. Consequently, if this newly infected applet were downloaded to another user's computer, its signature information would be invalid and it would be prohibited from running. The only way such a virus could spread would be to target Java developers' computers. If the virus could somehow infect Java applets *before* they were signed, then it might have a chance of spreading. Luckily, this is an unlikely scenario.

In general, Java viruses present little risk to the web-surfing public. Specifically, if a user downloads an unsigned Java applet that has been infected by a Java virus, they are at no risk since unsigned applets cannot access files. On the other hand, if the virus comes in as a signed applet, there is a higher degree of risk. If the user gives the applet access to his/her local system, then any number of attacks are possible, including virus attacks. However, as we have seen and will see in the next section, even with full access to a computer system, Java viruses are unlikely to spread.

Browsing-Serving Asymmetry

Today most end users don't run their own web sites directly from their home computer. So while a Java virus might find its way onto an end user's computer, it will quickly find that it has nowhere to

spread (beyond, perhaps, the browser's cache directory). The typical end-user's computer has virtually no Java applets that the virus could infect, since the typical end-user doesn't run a web-site on the computer they surf the web from. This will limit the ability for a mobile code virus (Java, ActiveX or HTML script-based) to spread. We call this phenomenon the *browsing-serving asymmetry*.

While this asymmetry will limit the spread of Java viruses today, Java virus infections could well grow in the future as more users obtain continuous cable or DSL-based connections to the Internet. When continuous connections become the norm, users will most likely use their machines to both surf the web *and* host their own web sites. As a result, the user may have a number of Java applets on his/her computer; this will provide a Java virus with a healthy supply of files to infect.

Consider what would happen if such a user browsed a web page that contains a viral Java applet. This applet would be downloaded, run on user's machine and could locate and infect a new Java applet that was part of the user's home page. If a second user then browsed the just-infected home page (i.e. downloads a newly-infected Java applet), this second user will then have their web page infected, and so on. In a sense, mobile code viruses are ahead of their time – their ability to do widespread damage is more likely to be brought on by advances in technology rather than by any further "enhancements" to the viral code by humans.

ActiveX and Netscape Plug-ins

Let's examine how ActiveX components are used by the client computer in a typical web browsing situation:

1. The web browser connects to a web-site server, for instance www.activexthreat.com.
2. The web browser downloads the appropriate web page, for instance *main.html*; the web page contains a reference to an ActiveX file or an archive .CAB file which can contain multiple pieces of an ActiveX component.
3. The web browser checks to see if the ActiveX component is already on the client system. If the ActiveX component is already on the client system (it was downloaded in an earlier session), then skip to step 7.
4. The web browser downloads the appropriate ActiveX file or CAB file.
5. The web browser checks to see if a digital certificate is attached to the ActiveX component.
6. Based on the security settings of the browser, the validity of the digital certificate and possibly the client user's input, the ActiveX object is either accepted and installed, or rejected.
7. The web browser runs the ActiveX component as specified by the web page (e.g. main.html). The ActiveX component is a 32-bit Windows program which has the same control of the computer as any Windows program.
8. Once the component has been downloaded and installed, the ActiveX remains on the client machine and doesn't need to be downloaded or re-installed when it is used by other web pages in the future.

Now that we have an understanding of how ActiveX content is used during web surfing, let's examine its potential as a carrier malicious mobile code.

ActiveX and Netscape plug-ins are the most dangerous of all mobile code.

That said, the Symantec AntiVirus Research Center has not received a single report of an actual ActiveX/plug-in malware or virus attack. This is a case of potential threat vs. actual threat. The potential threat from ActiveX/plug-ins is great, yet the actual threat has been non-existent.

ActiveX components are basically 32-bit Windows programs. When a user surfs to a web page which has one or more ActiveX components, these components are downloaded and run on the user's computer. Once an ActiveX component has been accepted on your computer, it has no restrictions like

those enforced by Java; it has free reign of your machine. Also, unlike Java applets, once an ActiveX component has been downloaded and installed on your computer, it remains installed until it is explicitly removed. This means that a browsing session can expose a computer to danger long after the browsing session has ended.

Netscape Plug-ins are Windows programs which are designed to integrate directly with Netscape Navigator. Like ActiveX, once they are installed, these plug-ins have access to the entire computer system and can perform any number of malicious actions. Unlike ActiveX, Netscape plug-ins do not have any code signing facility, so the user cannot verify who or where a plug-in came from. Luckily, Navigator will always prompt the user before installing such a plug-in. However, once a plug-in is installed, Navigator will not warn the user when its functionality is used or exploited by a web-page.

Signed vs. Unsigned ActiveX

In order to stem the threat of malicious ActiveX code, Microsoft has vigorously promoted code signing for ActiveX. A signed ActiveX component is one which is accompanied by a digital signature or certificate information, allowing the client to determine the origination and authorship of the component. The digital signature allows the web browser to certify that the ActiveX component's binary contents match 100% with the ActiveX component originally signed by the software provider (this is done via a hashing function). Depending on the security settings in the browser, the administrator (or end-user) can allow such signed ActiveX components to either have no access or **total access** to the client machine. For more information on code signing and digital certificates, see the *Digital Signatures* section below.

With regards to Internet surfing, Microsoft Internet Explorer offers the following default security settings for ActiveX:

1. Refuse all *unsigned* ActiveX by default.
2. Allow all *signed* ActiveX components, prompting the user before allowing one to be run.
3. Allow JavaScript and VBScript programs to use functionality from all *signed* ActiveX components that are marked *safe for scripting*.

If the user chooses to allow a new ActiveX component onto the system, that component will be installed permanently. If the user visits the same web-page at a later time, they will not be prompted to give access to the ActiveX component; access will be granted automatically. Furthermore, if when granting access to a signed ActiveX component, the user selects "Always Trust Content," then all subsequent ActiveX components from the signer in question will be downloaded and given full access to the system without further prompting or notification (during this and subsequent browsing sessions).

The default security policy in Internet Explorer puts the end-user in the driver's seat; the user is prompted to decide whether or not to allow ActiveX onto their machine. This is obviously a less than desirable in the corporation (or at home), since end-users are rarely able to decide whether or not content is harmful or not. Luckily, Internet Explorer allows the administrator to pre-configure settings and take such choices away from the user.

Malicious Mobile Code UsingActiveX and Plug-ins

ActiveX and plug-ins can be used to implement virtually any type of malicious code. Once either of these components has been downloaded and installed onto the client computer, it has full access to the machine.

It is interesting to note that we have seen very few denial of service attacks with ActiveX (in stark contrast to the number of these attacks we've seen in Java). In fact, the only such attack that the author could locate was the ActiveX *Internet Exploder* control. This ActiveX component shuts down the client system when a tainted web page is visited. It is certainly possible to implement denial of service attacks

with ease in ActiveX; however, virtually every proof of concept attack we have seen attempts to perform system modification or invasion of privacy. Why have we seen so few denial of service attacks implemented in ActiveX?

The scarcity of these attacks can be attributed to sociological rather than technological reasons. These attacks are simply too easy to program, and at this point in time, virtually all mobile malware code is being built as proof of concept. Just as Einstein would not be interested by Newton's apple-dropping experiment, malware authors have little or no interest in producing trivial denial of service ActiveX components. However, if and when hostile companies or foreign nations decide to use ActiveX as an offensive medium, we can expect to see this type of attack grow in prevalence.

Viruses

Given that one can implement virtually any type of malware with ActiveX and Netscape plug-ins, it is certainly possible to create an ActiveX or plug-in-based virus. However, to date, we have seen no ActiveX or plug-in viruses of any type.

While we have seen no such viruses, it is important to note that both ActiveX and Netscape plug-ins are essentially standard Windows programs and both can be infected by dozens of existing Windows viruses. If an ActiveX component were infected with such a virus, once downloaded, the virus could spread to other Windows executables on the client's computer, to the local area network and beyond. This is the most likely way that a virus could gain entrance to a client computer via ActiveX or a Netscape plug-in.

Like Java viruses, a *native ActiveX* virus would probably have a hard time spreading in today's computing environment [x]. There are two reasons for this; the first has to two with the *browsing-serving asymmetry* discussed below. The second has to deal with the nature of signed ActiveX:

If a native ActiveX virus were to locate another signed ActiveX component and modify it, then the just-infected-component's binary contents would no longer match the contents of the digital signature. Consequently, if this newly infected component were downloaded to another user's computer, its signature information will be invalid and it would be prohibited from running. The only way such a virus could spread would be to target ActiveX developers' computers. If the virus could somehow infect ActiveX components *before* they were signed, it might have a chance of spreading.

ActiveX and the Browsing-serving Asymmetry

Today most end users don't run their own web sites directly from their home computer. This means that while the end-user may have many ActiveX components to infect, no other web-surfer will ever run into these infected components. What happens if the end-user downloads an ActiveX virus? The virus would immediately infect all of the ActiveX components on the computer. Then where does it go? Nowhere. This will limit the ability for an ActiveX virus to spread. We call this phenomenon the *browsing-serving asymmetry*.

While this asymmetry will limit the spread of ActiveX viruses today, ActiveX virus infections could grow in the future as more users obtain continuous cable or DSL-based connections to the Internet. When continuous connections become the norm, users will be more inclined to use their machines to both surf the web *and* host their own web sites. Consequently, if a user downloaded and ran an ActiveX virus, it would infect a number of ActiveX components – some of which might be part of the end-user's own web page. If other users then connect to this user's web-site, they may inadvertently download infected ActiveX components, spreading the virus.

Safe For Scripting

When programming a new ActiveX component, developers can mark their component as *safe for scripting*. If an ActiveX component is marked as safe for scripting, once it is installed on the client's computer, any JavaScript or VBScript programs can call upon its functionality from within the web browser without any security restrictions. Such safe for scripting components are very useful since they permit the less powerful VBScript and JavaScript languages to implement many complex features that could only be implemented by a complex ActiveX (Windows) logic.

There is one big problem with ActiveX components that are marked safe for scripting. Should an end-user trust an ActiveX developer to properly "bullet-proof" their ActiveX control? Once such a control is installed in the client's system, *any* JavaScript or VBScript from any web site can use its functionality in any way it likes!

For instance, what if a developer were to build a safe for scripting ActiveX component that allows the user to download a graphics file, save it to any filename they liked and then view it? This component seems safe enough if used by non-malicious JavaScript. But malicious JavaScript might be able to use this component to save a graphics file over the C:\AUTOEXEC.BAT file. Or a malicious VBScript program could use it to save a corrupted image over the windows background graphics file. Either of these attacks could cause the machine to crash during startup, and far more malicious attacks are possible!

In general, marking an ActiveX component safe for scripting is a very dangerous proposition. Unless the component was impeccably designed, it may have security vulnerabilities and could be exploited by malicious or even just buggy JavaScript or VBScript.

JavaScript and VBScript

JavaScript and VBScript are programming languages which provide robust functionality to web pages. Web page authors can embed JavaScript and VBScript programs directly into their web pages to animate graphics, check fields of data submitted via web page forms for correctness, etc. While these script languages were designed to provide only limited functionality, researchers have found a number of vulnerabilities which allow malicious code to be written. **As of the time of this writing, all of the known holes have been plugged, making JavaScript and VBScript largely safe mobile code mediums, if the web browser is configured appropriately. It is important to note that JavaScript/VBScript can be used in conjunction with ActiveX and can exploit your computing environment!**

Signed JavaScript in Netscape Navigator

Like Java applets, JavaScript programs can be signed in Netscape Navigator version 4.0 and above to ensure security. If a web surfer allows a signed JavaScript program onto their computer, this program can request additional access to the computer and escape the JavaScript sandbox (Which is similar to the Java sandbox). In order for signed JavaScript programs to obtain more access to the computer, they must make use of Java via Netscape's LiveConnect technology. LiveConnect is a communications mechanism that can be used to allow different types of mobile code to work together; in this case, Java and JavaScript. Unlike traditional JavaScript which is embedded directly in web pages, *signed JavaScript* is delivered inside of a .JAR archive file and are linked to from the base web page.

JavaScript and VBScript Back Doors

As discussed in the ActiveX section of this document, it is possible for ActiveX programmers to specify that their components are *safe for scripting.* Once an ActiveX component has been marked in this fashion and installed on a client computer system, its functionality can be used by any JavaScript or

VBScript programs without any security notifications to the user. Furthermore, these JavaScript/VBScript programs can come from any web-site and still use the features of the ActiveX component; the scripts do not have to come from the same web-site as the ActiveX control to make use of it.

This poses no risk if the safe for scripting ActiveX components installed on the client computer have been appropriately tested and secured. However, if the components are not properly designed or are buggy, they can be exploited by malicious or poorly designed script logic. In essence, by installing an ActiveX component marked safe for scripting, Internet Explorer users are potentially opening up a back door into their system.

JavaScript, VBScript and Malware

As with the other mobile code attacks, the Symantec AntiVirus Research Center has no recorded JavaScript or VBScript malware incidents to date. However, a number of proof of concept attacks have been demonstrated since the introduction of these scripting languages.

Researchers at the Open Research Institute (www.opengroup.org) and Bell Labs have uncovered a number of holes in the JavaScript language. The researchers built a JavaScript program which could monitor web usage, even after the user left the page that delivered the malicious script. This script could monitor all visited URLs, the values of fields that were filled in (with contents such as passwords), etc. The same researchers discovered other vulnerabilities, including one which allowed the script to send a directory listing of the client's files to an attacker. Finally, the group found that JavaScript could be programmed to forge e-mail messages from the client user.

Today, all of the known holes have been plugged and new holes are being found at a much slower rate. This would imply that most of the holes in these two systems have been located and removed, but there are no guarantees. It is important to note that both JavaScript and VBScript can be used to access files and other resources on the client computer – if given permission to do so by the user; however, browsers can be configured to automatically deny this access.

JavaScript and VBScript Viruses

To date, two VBScript viruses (which work under Internet Explorer) have been written (there are no known JavaScript viruses to date). The first such virus is written in VBScript and embedded at the top of an infected web page. When an infected web page is downloaded, the virus runs and attempts to locate other HTML pages on the client computer. If it finds such a HTML file, it will prepend itself to the new web page. When trying to get access to the files on the hard drive, the virus triggers Internet Explorer's security [xi] and the user is asked if they would like to allow the action to continue. If the user says no, the virus will fail. Otherwise, the VBScript will locate other web pages on the local drive, if there are any, and infect them.

While this virus cannot spread without the express permission of the user, it still poses a security risk. Users aren't prompted "Do you want the virus to run?" Instead, they're prompted with a message like, "Do you want to allow this VBScript program to access files." Such a question might have an obvious answer to the power user, but will make little if any sense to the average end-user. Consequently, JavaScript and VBScript should be used with care in the corporation and the home. Fortunately, corporations can configure VBScript to automatically deny such operations without the consent of the employee.

Word and Excel Macros and Browsers

Recently, there has been a great deal of news about Word and Excel mobile code attacks. The "Russian New Year" attack, described by Finjan Software, is one such attack which was received attention. The basic idea behind these attacks is as follows:

1. A user has Internet Explorer and Microsoft Office installed on their machine.
2. The user surfs to a web page that has an embedded Microsoft Office document (such as a Word or Excel document) with macros.
3. Under certain configurations, Internet Explorer will automatically download the embedded document/spreadsheet and run its macros or formulas without any notification to the user.
4. These macros/formulas run with the same privileges as the user and can delete files, send e-mail, open up Internet connections, etc.

In a paper authored by John Morar and David Chess at the 1998 Virus Bulletin Conference, the authors described how a typical installation of Internet Explorer and Microsoft Office could cause a computer to be susceptible to such an attack [xii]:

1. Install Windows 98.
2. Install Microsoft Office 97 Professional, Service Release 1, and accept all of the defaults for a typical installation. While using Word on local files, click on the check box that offers to stop reminding you every time you open a local Word file which contains macros. (Note: The Russian New Year exploit will work without prompting the user even if the user does not disable the macro reminder feature in Excel!)
3. During the office installation, PowerPoint, Excel and Word were registered for immediate execution, so when a user clicks on a link to a Word document, you are only depending on Word itself to warn you about macros, however, that warning was turned off in the previous step by a user who was tired of being reminded every time he opened a macro-containing file locally. Now, documents behind Web links are opened in Word without any prior notification.

Note: Internet Explorer can be configured to prompt before execution, by checking 'Confirm open after download' for each of the major file types (see *Windows Explorer*: View → Folder Options → File Types.)

These attacks are documented and real and should be considered a potential threat. However, as with virtually every other form of malicious mobile code, we have never seen this hole exploited for an actual attack. It has only been demonstrated as a proof of concept by researchers.

Anti-malware Technologies

Signature Scanning

Signature or "fingerprint" scanning is the oldest and most mature of the anti-virus technologies. In general, signature scanning can only be used to detect known, pre-analyzed malware threats and is unable to detect new or unknown threats.

To discuss the effectiveness of fingerprint scanning, let's segment malware threats into two different groups: *immediate payload threats* and *delayed payload threats*.

Immediate payload threats are those pieces of malware which *drop their payload* or cause harm almost immediately. Harm can be caused in any number of ways, not just data destruction:

1. System modification attacks, including destruction of data, data diddling, etc.
2. Invasion of privacy attacks, including sending information or user files over the web.
3. Denial of service attacks, such as crashing the machine, causing network congestion or bringing the network down (worms)

Most Trojan horse programs and proof of concept ActiveX and Java malware can be considered immediate payload threats because they do their damage as soon as they are invited onto a system and executed. Also, some worms can also be considered immediate payload threats since they may overly congest or bring down the network in trying to spread, whether or not they have a secondary payload.

In contrast, delayed payload threats are those malware programs that do not do significant damage to computing systems or do their damage after an extended period of time following initial infection. Many computer viruses fall under this category; they may infect a system and spread from file to file but will not cause significant damage immediately (until, for instance, March 6th in the case of the Michelangelo virus).

Signature scanning can be an effective technology for detecting and removing delayed payload threats if the payload delay time is longer than the time it takes anti-virus researchers to obtain a sample of the threat and distribute a signature. Signature scanning is useless against immediate payload threats and delayed payload threats that deploy their payload in a timeframe which is smaller than the response time of anti-virus researchers.

If we assume that most mobile malware will be of the immediate payload variety, then signature scanning technology will provide little value to corporations and end users against this threat. If we assume that most mobile malware will implement delayed payloads, then signature scanning can be an effective protection mechanism.

Given the dearth of real-world mobile code attacks, it is difficult to determine how well this solution will protect users. However, with recent examples such as the PICTURE.EXE Trojan, it would seem that Trojan horses and other mobile code can gain large circulation and cause large amounts of damage well before anti-malware vendors are able to analyze the mobile code threat and respond.

Heuristics

The heuristic scanner works by examining the program logic of executable files, document macros, or diskettes to determine whether or not their program logic is capable of exhibiting viral or other malicious behaviors. If the program logic appears to be malicious, the anti-malware product can alert the user to the potential threat.

While many anti-virus products can detect a high percentage[xiii] of new virus threats, it is not clear that heuristic technology can be successfully adapted to other, more general types of malware. Viruses are easy to detect heuristically because the vast majority infect files (disks) in very specific ways. Consequently, it is easy to isolate the likely regions where viruses will be hiding and perform extensive code analysis on these areas.

This task is much more difficult for generalized malicious mobile code. Unlike viruses which tend to localize their malicious code (at the top or end of a file, for instance), general malicious mobile code can have its logic distributed over megabytes of a the booby-trapped file. This makes the timely code analysis that is required by heuristics extremely difficult, if not impossible. Also, the high level nature of many malicious programs (with the exception of viruses) also makes these threats difficult to detect heuristically. If the malware is written in a high level language, it's malicious logic is not only distributed through the executable file, but will also be much harder to isolate and identify. Hand-written assembly language routines are extremely easy to analyze because they are purposeful in their actions. Assembly language generated from high level languages is for more obscure and difficult to reverse engineer.

Also, given that malware authors can obtain the same anti-malware products used by their targeted victims, these authors can also "tweak" their malware creations to avoid detection by these products.

Import Scanning

For a malicious virus, Trojan, Java or ActiveX applet to propagate itself or do damage, it must call upon the operating system. For instance, it must ask the operating system to delete files, to format the hard drive or to connect to a rogue web site and export data.

If a malicious Windows (ActiveX) program or Java applet does call on the operating system, it will have an internal list of all of the operating system functions that it requires in order to do its damage. This is called the *import list* and is found in every Java, ActiveX and Windows program file. For instance, a cute graphics applet might use the following operating system functions: "DrawCircle," "ShadeCircle," "ShowWindow." On the other hand, a rogue applet might use "DrawCircle" and "DeleteFile." DrawCircle would draw a cute circle on the screen while DeleteFile deleted all the user's files.

Import scanning products work by examining the list of operating system functions that an applet or program needs in order to do its work. These products have *profiles* of what types of functions are allowed and which are disallowed in the enterprise. If all of the functions referred to by an applet/program are on the approved list, the software is allowed into the enterprise. If any of them are found on the unapproved list, the applet can be quarantined and rejected.

Import scanners are prone to both false positives and false negatives. For example, many useful programs/applets may need to delete files or open an Internet connection and will refer to these operating system functions in their import lists. Unfortunately, the Import Scanning software cannot determine the "goodness" or "badness" of an applet; it can only determine whether or not the mobile code uses operating system functions that can potentially do bad things.

Unfortunately, malicious applets can be designed to conceal their reliance on the operating system. Many of today's 32-bit Windows viruses call upon the operating system in obfuscated ways; the import tables in infected files do not refer to any suspicious operating system functions. Yet these programs are fully capable of infecting other files. Import Scanners will fail to filter such mobile code, allowing them to wreak havoc in the enterprise. Luckily, it is more difficult to hide reliance on the operating system in Java programs; so this technology may be more effective at protecting against malicious Java applets. However, Java is the safest mobile code platform already!

Digital Signatures

Both Netscape Navigator and Internet Explorer support digital signatures and certificates to protect against malicious Java code, JavaScript and ActiveX components downloaded over the Internet[xiv]. In addition, a number of gateway products also provide ActiveX and Java signing functionality. Each time an applet or ActiveX control is downloaded from a web site, it may be accompanied by a certificate. This certificate can be used to *authenticate* the origin of the applet and verify that its content has not changed since the developer distributed it.

Web browsers can be configured to prompt the desktop user each time they encounter a signed applet and allow them to accept or reject the applet. Alternatively, the browser can be configured to automatically allow or disallow either applets which are unsigned, or applets which are signed by providers on an accepted list or a rejected list.

If a web surfer encounters a signed ActiveX component on the web, the surfer has two choices: to refuse the component or to allow it onto the system with the full security privileges of the surfing user. If the user can open document files, the ActiveX component can; if the user can change the registry, so can the component. In contrast, Java code signing provides more granular security. Signed Java programs can be given access to some system resources and denied access to other resources. For example, Internet Explorer provides the following YES/NO options for signed Java applets:

1. Ability to manipulate groups of applets (threads) running on your computer.
2. Ability to accept connections from other computers on a network.

3. Ability to load restricted Java system code.
4. Ability to display windows that don't have the unsigned applet label.
5. Ability to manipulate other applets (threads) running on your computer.
6. Ability to use native code stored in dynamically linked libraries.

Digital signatures effectively protect against *stationary* types of malware since they allow the client to determine exactly who produced the content in question (in fact, digital signatures *create* stationary content). The liability introduced by digital signatures is a strong motivator for the applet provider to deliver non-malicious software. Unfortunately, digital signatures can sometimes lend a false sense of security. If a trusted vendor inadvertently infects an ActiveX component with a Windows virus and then signs this infected component, an unsuspecting web surfer may unwittingly contaminate their system with a virus.

URL and Attachment Blocking

URL Blocking technology prevents users from surfing to WWW sites that have potentially malicious content. This technology can block sites based on their URL, their IP address or the textual content of their web pages (for instance "get the latest Back Orifice Trojan here"). This technology is typically implemented on an HTTP proxy or directly on the firewall in corporations; this allows for easy centralized policy management and reduces the costs of individual desktop configuration. Alternatively, this technology can be deployed on each desktop, but this will result in higher maintenance costs (for instance, to update the list of blocked sites).

File and attachment blocking is a technology which can be implemented at the HTTP gateway, the e-mail gateway or at the groupware server. This technology strips all possible malware content from web traffic and e-mail attachments before the content reaches users. All executable programs can be removed from the e-mail or filtered from web pages, while document macros can be stripped, leaving the accompanying document intact. This stripping process should be applied to content coming into the enterprise from the Internet as this content is less likely to be safe. Depending on the level of blocking – blocking all executable content vs. just suspicious content - such a solution can cause varying amounts of inconvenience for employees; however, blocking executable content is one of the most effective ways to ensure a safe computing environment.

Behavior Blocking

When installed, the behavior blocker integrates into the operating system on the client machine and watches for malicious program behaviors in real time. As we have seen, all malicious applets must utilize the operating system in order to perform their malicious actions. The behavior blocker intercepts these requests before they can reach the operating system and alerts the user or administrator. Since the behavior blocker integrates directly into the operating system, it can trap a wide variety of operating system requests regardless of whether or not the mobile code attempts to obfuscate its use of these functions in its code; eventually the malicious code must interact with the operating system and its actions will be caught.

The exception to this rule is *tunneling malware;* this term is used to refer to malware which specifically attempts to bypass behavior blocking technology. A small number of DOS viruses already employ these techniques, although no other malware is known to use this technique.

Behavior blocking technology can be used to protect against virtually all types of malware. There are two major drawbacks to this technology:

1. The malicious mobile code must actually be running and trying something harmful in order to be caught and terminated.
2. Some actions carried out by programs may not be malicious, yet appear malicious. Like the *import scanner,* the behavior blocker has a hard time distinguishing good behavior from bad.

Consequently, existing behavior blockers tend to generate false alarms and can be obtrusive to users.

While most behavior blockers have been implemented as desktop-based solutions, at least one vendor has created a gateway-side behavior blocker for Java. As Java programs are downloaded from the web by end-users, they are detoured to a high-powered Java server. The applets are allowed to run on this server and all I/O is provided to the user so they receive the same web experience. If a malicious applet attempts to access any data or cause harm, these attacks are restricted to the Java server, protecting data on the corporate desktops and servers.

Seven Suggestions for Safeguarding Your Corporation

Each of the anti-malware technologies described above can help to prevent malware from becoming a liability to the enterprise. How should your organization use these pieces to yield the highest level of security with the least amount of intrusiveness for your users. A seven-part answer to this question is presented below; each additional suggestion yields more robust malware security; however, each successive recommendation may also result in a more obtrusive solution for you and your users. By determining the security needs of your organization, you can use these suggestions to implement an effective malware strategy.

Run Anti-malware Software On All Desktops, Servers and Gateways

It is critical to run anti-virus (anti-malware) software on the desktop, at the gateway (e-mail or firewall), on file servers and on groupware servers. Most anti-virus solutions also detect and remove known mobile code threats in addition to viruses and Trojan horses. In addition, anti-virus software has had the longest time to evolve. Most anti-virus products use a number of technologies (such as those described above) in concert, enabling these products to detect the widest variety of threats. Many anti-virus products employ signature scanning, heuristic scanning and behavior blocking technologies.

A multi-tier anti-malware solution will protect your users against a huge number of existing threats and limit your organization's vulnerability to some new and unknown threats. Remember, it's critically important to keep all of your anti-virus signatures files and engines as up-to-date as possible and consistent across all platforms to protect against the latest malware threats!

Install URL blocking software at the gateway or the desktop

URL blocking software can be used to prevent your users from surfing to a number of productivity-draining sites. It can also prevent your users from going to known Vx (Virus Exchange) sites and downloading viruses for experimentation. In most cases, URL blocking software will not hinder your employees from doing their work and will prevent some infections by malware or viruses.

Unfortunately, URL blocking software is only useful for blocking known malware sites or sites which specifically indicate that malware is available (if content scanning/blocking is employed, the software can check for text like "get viruses here" in the web page and block the site). This means that URL blocking will not help to protect against:

1. Web sites which are not known malware distribution centers and that intentionally post malicious attacks to harm users.
2. Web sites which inadvertently post malware without knowing it.

Only Allow ActiveX From A Limited Set Of Authenticated Providers

If your users do not need to use ActiveX or Netscape plug-ins to do their work, it makes sense to configure your corporate web browsers or HTTP proxy to reject **all** ActiveX controls or plug-ins. As we have seen, ActiveX components can perform any number of malicious attacks and should be considered very dangerous. While liability issues (associated with stationary threats) may prevent ActiveX from becoming a major threat, eventually some attacker will weigh the profit of building and deploying this malware vs. the potential liability of getting caught and initiate an ActiveX attack on end users or corporations.

If disabling all ActiveX is too restrictive for your users, you can also configure Internet Explorer[xv] to only allow properly *signed* ActiveX controls. Establish a short list of trusted partners and distribute their digital certificates to your desktops or to an HTTP proxy that can filter improperly signed ActiveX. This will allow your employees to use a limited set of trusted ActiveX and greatly reduce the risk of an attack. Also, configure your desktop browser software to prevent it from allowing users to add new trusted signatures; this will ensure that your users don't inadvertently invite signed malware from an untrusted source into the enterprise.

If you use Netscape Navigator, you may want to configure Navigator to refuse all new plug-ins; otherwise, your users may add their own plug-ins during web surfing which may open additional security holes in your computing infrastructure. For instance, a Netscape plug-in is available that allows your users to use ActiveX content in Navigator (Navigator does not support ActiveX on its own). However, while this plug-in does support ActiveX, it does not support code signing or digital certificates. Consequently, if your users install such a plug-in, they'll be opening your network up for ActiveX-based attacks from both signed and unsigned malicious components!

Java, JavaScript and VBScript Suggestions

With properly configured web browsers, unsigned Java, JavaScript and VBScript are very safe platforms and can be used in the enterprise without significant risk to the corporate computing infrastructure.

While unsigned Java applets and other script-based languages such as JavaScript run in a sandbox and represent little risk to the enterprise, signed Java applets can request access the system and wreak havoc just like ActiveX components. If your users do not need Java applets that have access to the local computer in order to do their work, configure your corporate web browsers or HTTP proxy to automatically terminate/reject all Java applets that request additional access to the local computer. This will take the decision-making power away from your users and ensure that all Java applets running in the enterprise run in the safety of the Java sandbox.

If this measure is too restrictive for your users, you can also configure the major browsers to only allow properly *signed* Java applets from a set of trusted partners. Establish a list of trusted Java-providing partners and distribute their digital certificates to your desktops or to an HTTP proxy. In this way, properly signed applets, and only these applets, will be allowed to access your local systems' resources, reducing the risk of an attack.

It's a good idea to configure your desktop browser software to prevent it from allowing users to add additional trusted signatures on their own. Also configure the desktop browser software to automatically deny all requests by JavaScript, VBScript, etc. to access the local machine resources. **By default, the major browsers will prompt the user asking them if it is ok for the script to access the local computer.** This decision should not be made by your employees; configure your browsers to deny these requests automatically.

On those machines which have proprietary or mission critical information, we recommend that all programmable content (and in fact all web browsing) be disallowed. If these machines are on the corporate

network, running a personal firewall on these boxes will also help to reduce the risk of penetration to infinitesimally low levels.

Computers with Java development environments (and a *CLASSPATH* containing additional Java components) can potentially be susceptible to attack by malicious Java applets. If a programmer (or Java development environment) installs unsafe Java components into directories specified in the CLASSPATH, this can open the computer up to Java-based attacks. What is the likelihood of this? If the Java components in the CLASSPATH on such a computer are all proprietary and not known to the general (and hacking) population, the likelihood is that an attacker will have a very low probability of locating and exploiting these holes. However, if a common Java component installed by a Java development environment contains such a hole, it may be easily exploited. Here's a recommendation: If your organization is extremely worried about security, isolate all machines with Java development environments from the web (other suggestions: write a "shell" program that clears out the class path while a Java enabled browser runs, have dual-boot machines with the Java development environment on one configuration and web-surfing apps on the other.).

Obtain the latest patches for your web browser and e-mail products

Over the last few years, researchers have discovered a number of security holes in Internet Explorer, Netscape Navigator and popular e-mail programs. Luckily, the rate at which holes are discovered is slowing, indicating that these products are reaching a level of acceptable security. In most cases when a hole is found, the discoverer works with the product vendor to remove the vulnerability as quickly as possible.

While we have never encountered "in the wild" exploitation of any of the holes described in this paper, it's a good idea to keep up-to-date on these security holes by visiting your vendors' web sites and to install the latest patches to your Internet-based software (We understand that it is costly to install new updates, but it will help to ensure security). Alternatively, for those with proxy-based firewalls, it may be possible to plug these holes on the firewall. Contact your firewall vendor for more information.

For security advisories, consult the Computer Emergency Response Team, a non-profit organization run at Carnegie Mellon University: http://www.cert.org/ Also check http://www.sarc.com/ for information on the latest malware threats.

Install Software To Filter Executable Files/Strip Macros From Incoming E-mail and HTTP Traffic

SARC, the ICSA and others have found that the vast majority of all viruses and Trojan horses are delivered via e-mail; while Trojan horses and viruses received in e-mail may not strictly constitute malicious *mobile code*, they now constitute the majority of malware incidents in the enterprise. Consequently, filtering incoming e-mail attachments that may harbor viruses or Trojan horses will help to solve the very real problem of malware in the enterprise.

Some gateway anti-virus solutions can be configured to strip all incoming executable files and/or strip the macros from incoming documents. This drastic measure may impact your users by denying them access to needed executable or macro content. On the other hand, it will virtually neutralize the Internet as a source of malware given the current landscape of the threat.

Conclusions

Mobile code is becoming an increasingly important component of the Internet experience and in many cases *must* be used to provide appropriate functionality for Internet applications. Today's mobile

code systems – Java, ActiveX, and the various scripting languages – provide varying levels of security and varying levels of risk for the enterprise. Security flaws and holes have been discovered in virtually every single mobile code platform, making the problem even more confusing.

Protecting the enterprise from viruses is a full-time job and adding mobile code to the landscape only complicates matters. Unfortunately, many of today's computing systems are designed with functionality and not security in mind. An important first step to providing effective security for your organization is determining the required level of security and how much this will impact your users; this paper lists a number of increasingly secure (and increasingly obtrusive) security suggestions for protecting against mobile code.

After understanding the security requirements for your corporation, the suggestions above will help you deploy the proper products and policies to achieve a safe computing environment with respect to mobile code. While written policies will have some impact on security, enforceable, software-based solutions are ultimately the best way to keep your users productive and your machines healthy.

The good news is that there are no known mobile code attacks that have actually targeted end-users of corporations. The bad news is that some of the mobile code platforms do have security holes which make such attacks possible. Remember our "running with scissors example;" it's not a safe thing to do, but it rarely results in catastrophic damage. Likewise, mobile code can provide an equivalent threat to your computing environment, although in practice (at least today), it is more of a potential threat than a real threat.

Good luck and safe computing – and don't tease the (digital) animals!

---

[i] Based on all of the author's research, not a single actual attack has been documented with Java or ActiveX. While there are hundreds of malicious "proof of concept" mobile code attacks, the author is unaware of anyone who has been exploited by such an applet.

[ii] We have one recorded stationary incident as of the writing of this paper, which did not use ActiveX or Java, but a standard 32-bit Windows program. Ghosh, Anup, *E-Commerce Security, Weak Links, Best Defenses*, 1998, pp 18-19.

[iii] McGraw, Gary and Felten, Edward, *Java Security, Hostile Applets, Holes and Antidotes*, 1996.

[iv] See http://www.symantec.com/avcenter/venc/data/picture-exe-th.html for more information.

[v] Key grabbers install themselves into the operating system and record all keystrokes. This keystroke list can then be sent to an attacker to obtain passwords and other private information. Even if a user uses a secure Internet connection, his or her keystrokes can still be grabbed and sent to an attacker.

[vi] Ghosh, Anup, *E-Commerce Security, Weak Links, Best Defenses*, 1998, pp 73-75

[vii] In searching the web, the author could only find one new invasion of privacy or system modification attack discovered in 1998.

[viii] McGraw, Gary and Felten, Edward, *Java Security, Hostile Applets, Holes and Antidotes*, 1996, chapter 2, section 2. This list is a partial list intended for the administrator rather than the programmer.

[ix] When a software developer digitally signs a Java applet, a cryptographic check-sum is taken of the Applet's binary contents. When the applet and its signature is later downloaded to an end-user's machine, the browser computes a checksum of the contents of the applet and compares this to the original checksum. If the two match, it is clear that the applet has not changed during transit and is as the software developer originally intended.

[x] Definition: A native ActiveX virus is one which is implemented in ActiveX and which only spreads to other ActiveX components.

[xi] The default Internet Explorer settings will prompt the user to approve the operation. More stringent settings can be used to prevent the user from making this decision.

[xii] John Morar and David Chess, 'Web Browsers –Threat Or Menace?', Proceedings of the Virus Bulletin International Conference; Munich Germany; October 1998.

[xiii] Industry estimates are that 70+% of DOS viruses can be detected heuristically, 90% of all macro and boot viruses can be detected heuristically as well.

[xiv] Netscape Navigator has digital certificates for JavaScript, Internet Explorer has digital certificates for ActiveX. Both support digital certificates for Java.

[xv] Netscape Navigator/Communicator does not support ActiveX. Users of this software can obtain a Navigator plug-in which allows ActiveX to be used; however, Navigator offers no code signing or certification for ActiveX. It is recommended, therefore, that ActiveX always be disabled under Netscape Navigator.