

# Active Content Security

Date: December 13, 1999

Authors: IS Tutorial Group 10-11  
Owen Brady  
Steven Clay  
Hilary Ganley  
Mansour Al-Meaither  
Lazaros Haleplidis  
Matthew Joyce  
Samuel Yee

<a href="#">1</a>	<a href="#">INTRODUCTION</a>	3
<a href="#">1.1</a>	<a href="#">ACTIVE CONTENT SECURITY</a>	3
<a href="#">2</a>	<a href="#">ACTIVE SCRIPTING</a>	4
<a href="#">2.1</a>	<a href="#">INTRODUCTION TO ACTIVE SCRIPTING</a>	4
<a href="#">2.2</a>	<a href="#">OR ECMASCRIPT?</a>	4
<a href="#">2.3</a>	<a href="#">UNDERSTANDING JAVASCRIPT</a>	4
<a href="#">2.4</a>	<a href="#">EXAMPLE USAGE OF JAVASCRIPT</a>	5
<a href="#">2.5</a>	<a href="#">KNOWN VULNERABILITIES OF JAVASCRIPT</a>	6
<a href="#">2.6</a>	<a href="#">NETSCAPE SECURITY POLICY</a>	10
<a href="#">2.7</a>	<a href="#">INTRODUCTION TO SCRIPT SIGNING</a>	11
<a href="#">2.8</a>	<a href="#">GRANTING OF PRIVILEGES</a>	12
<a href="#">2.9</a>	<a href="#">CONCLUSIONS</a>	12
<a href="#">3</a>	<a href="#">PLUG-INS</a>	13
<a href="#">3.1</a>	<a href="#">OVERVIEW OF PLUGIN SECURITY</a>	13
<a href="#">3.2</a>	<a href="#">KNOWN SHOCKWAVE PLUGIN VULNERABILITY</a>	13
<a href="#">3.3</a>	<a href="#">CONCLUSIONS</a>	15
<a href="#">4</a>	<a href="#">JAVA</a>	16
<a href="#">4.1</a>	<a href="#">HISTORY</a>	16
<a href="#">4.2</a>	<a href="#">JAVA LANGUAGE SECURITY</a>	17
<a href="#">4.3</a>	<a href="#">THE SANDBOX CONCEPT</a>	18
<a href="#">4.4</a>	<a href="#">THE COMPONENTS OF THE SANDBOX</a>	18
<a href="#">4.5</a>	<a href="#">DIFFERENT CLASSES OF SECURITY: TOWARDS JAVA 2</a>	21
<a href="#">4.6</a>	<a href="#">JAVA APPLET VULNERABILITIES</a>	21
<a href="#">4.7</a>	<a href="#">ATTACK APPLET USING TYPE CONFUSION</a>	24
<a href="#">5</a>	<a href="#">ACTIVEX</a>	25
<a href="#">5.1</a>	<a href="#">OVERVIEW</a>	25
<a href="#">5.2</a>	<a href="#">INTRODUCTION</a>	25
<a href="#">5.3</a>	<a href="#">ACTIVEX SECURITY</a>	27
<a href="#">6</a>	<a href="#">PROTECTION AGAINST HOSTILE APPLETS</a>	29
<a href="#">6.1</a>	<a href="#">TURN IT OFF</a>	29
<a href="#">6.2</a>	<a href="#">USE LATEST MAINTENANCE RELEASE OF BROWSER VERSION</a>	29
<a href="#">6.3</a>	<a href="#">FILTER AT THE CLIENT FOR KNOWN MALICIOUS APPLETS</a>	30
<a href="#">6.4</a>	<a href="#">FILTER AT THE FIREWALL FOR KNOWN MALICIOUS APPS</a>	31
<a href="#">6.5</a>	<a href="#">ROUTE ALL MOBILE CODE THROUGH SACRIFICIAL MACHINE “CAGE”</a>	32
<a href="#">6.6</a>	<a href="#">RING AROUND THE DATA:</a>	32
<a href="#">6.7</a>	<a href="#">RESOURCE LIMITING</a>	33
<a href="#">6.8</a>	<a href="#">APPLET SIGNING</a>	33
<a href="#">7</a>	<a href="#">REFERENCES</a>	35

# 1 Introduction

*To unfailing take what you attack, attack where there is no defense. For unfailingly secure defense, defend where there is no attack.*

*Sun Tzu*

Downloading a program from the Internet and running it is a dangerous activity. There are no limits to what a program can do once it is active on your machine. But Internet users are beginning to understand that downloading and running arbitrary programs may be a bad idea.

However most users do not think of Active Content in the same way. Visiting a web page and watching a piece of animation play on the page or playing a small game that seemed to arrive embedded in an email from a friend hasn't gained the same bad name as downloading programs of doubtful origin. However, this may change.

## 1.1 Active Content Security

This report investigates Active Content technologies, showing their security capabilities and vulnerabilities along with some examples of how those vulnerabilities have been exploited. In conclusion, there is a summary of the possible countermeasures against hostile Active Content.

We have defined Active Content as four classes of technology that enable anything from simple animation on a web page to full-blown distributed applications:

- Active Scripting languages – JavaScript & VBScript
- Browser plugins – Shockwave et al
- Java
- ActiveX

## **2 Active Scripting**

### **2.1 Introduction to Active Scripting**

A web page can never be interactive without the extensive use of Web Client Active Scripting. Active Scripting is defined here as a scripting language that able to control the behaviour and the content of the web browser window. The shorter name, Active Scripting would be used for the rest of this document. Active Scripting has a wide range of usage for the web browser. It is used primarily to respond to any user events such as mouse clicks, form validation, and navigation. Later versions of Active Scripting even allow HTML (Hyper-Text Markup Language) objects to be animated and displayed dynamically. This is indeed a pillar of the recent so-called Dynamic HTML.

Today, the most popular Active Scripting languages are the Netscape's JavaScript and the Microsoft's VBScript. Both competing Active Scripts exist for the same purposes and are doing the same things. However, this report would use only JavaScript for the sake of clear explanation. We should perhaps make it clear that JavaScript is related to Java by name only – any connection between the two exists only in the heads of the marketing folks.

### **2.2 Or ECMAScript?**

The archrival of Netscape, Microsoft, has also come up with a similar, but not identical, which called JScript. Therefore, both Netscape and Microsoft work with the ECMA (European Computer Manufacturers Association) and jointly formed a standard, namely ECMA-262. According to Microsoft, JScript 3.0 is fully compatible with the standard while JavaScript 1.3, according to Netscape, has adhered to it.

### **2.3 Understanding JavaScript**

Netscape defines JavaScript as a cross-platform, object-oriented scripting language. Unlike most object-oriented programming languages like C++ and Java, JavaScript does not have a compile-time system of classes. JavaScript supports a runtime system based on a small number of data types representing numeric, Boolean, and string values.

In addition, JavaScript supports functions without any special declarative requirements. Functions can be properties of objects, executing as loosely typed methods.

The execution of JavaScript could be driven by user-events such as mouse movement or form submission. Therefore, the necessary actions could be taken care by the event-handlers written in form of JavaScript function.

JavaScript has a prototype-based object model rather than the more common class-based object model. The prototype-based object model provides dynamic inheritance; that is, what is inherited can vary for individual objects. JavaScript objects could be created and described by programmers. More importantly, JavaScript also maintain a hierarchical set of pre-defined browser objects (e.g. Navigator Objects – window, frame, history etc.) to be manipulated by programmers.

For examples, JavaScript could “pop-up” some small friendly windows, submit a data form automatically, and re-forward to other URLs. Useful to provide unlimited interactive web experiences to surfers, however, it is also major security vulnerability for both hackers and crackers to exploit. In fact, most JavaScript vulnerabilities described by this document are based on these browser objects.

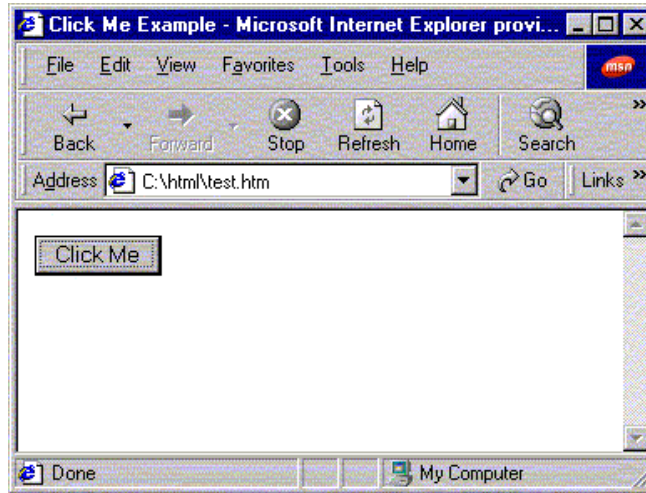
Though JavaScript, no matter how malicious, could not really crash the system or delete any files by itself, it has the every potentials and abilities to reveal precious user information and even steal any files and send to any remote servers. In another word, the confidentiality of the user system could be unknowingly compromised.

## 2.4 Example Usage of JavaScript

It is not the purpose of this document to teach JavaScript. However, it would be beneficial provide some quick reference on JavaScript. A simple example of HTML and JavaScript is described as follow.

```
1. <HTML>
2. <HEAD><TITLE> Click Me Example </TITLE></HEAD>
3. <BODY>
4. <FORM>
5. <!-- This will create a button -->
6. <INPUT type=button Value="Click Me"
   onClick=ClickEvent ()>
7. </FORM>
8. </BODY>
9. <SCRIPT language=JavaScript>
10.   <!-- Event Handler for mouse clicking -->
11.     function ClickEvent() {
12.       alert('You have clicked the button!');
13.     }
14.   </SCRIPT>
15. </HTML>
```

Line 6 will create and display a button as shown below.



On clicking the <Click Me> button, the event handler (ClickEvent) will be activated and executed. In this example, a dialog box will be displayed as shown.



## 2.5 Known Vulnerabilities of JavaScript

There are many known security loopholes on JavaScript. Much of such warnings had been brought up and issued by the World Wide Web Consortium (W3C) and many others. The following vulnerabilities are extracted and edited from W3C security FAQ. The respective browser vendors had fixed most security loopholes, but some do not have any patches yet.

### 2.5.1 Ability to Read Arbitrary Files on User's Machine (November 1998)

A bug in the JavaScript implementation in Netscape Communicator 4.5 and 4.04-4.05 allows a Web page to read arbitrary files from the user's machine and transmitted across the Internet. Any file that can be read with the user's permissions is vulnerable, including the system password file. The bug affects both Windows and Unix versions of Communicator. Any HTML page can carry this exploit, including ones that are transmitted as an e-mail enclosure. Internet Explorer has not been reported to be vulnerable. It is not known whether Netscape has provided any patches.

### **2.5.2 The "Cuartango" Hole (November 1998)**

Microsoft Internet Explorer is also vulnerable to file theft via JavaScript. Internet Explorer versions 4.0-4.01 and pre-release versions of IE 5.0 allow JavaScript programs to cut and paste text into file upload fields, thereby allowing a booby trapped Web page or e-mail message to steal any file on the user's disk. It is named "Cuartango" because Mr. Juan Carlos Garcia Cuartango from Spain discovered the security hole. Another two holes namely, Son of Cuartango and Grandson of Cuartango, were discovered shortly after. However, Microsoft was able to come up with all the necessary patches to fix the respective bugs by end of 1998.

### **2.5.3 The Netscape "Cache Browsing Bug" (October 1998)**

A hole in the Windows versions of Netscape Navigator 3.04, 4.07 and 4.5 allows remote web sites to read URLs from the browser cache, allowing them to intercept the list of sites recently visited by the user. Mac OS and Unix versions are not affected. No patch is currently available for these software versions.

### **2.5.4 Ability to Intercept the User's E-Mail Address and Other Preferences (February 1998)**

Versions of Netscape Navigator 4.0 through 4.04 contain a security hole involving access of JavaScript programs to the browser preference settings. The settings, which are stored in a file named preferences.js in the Netscape directory, include a variety of private information such as user's e-mail address, the names of mailbox files, and the identity of user's e-mail and news servers. In addition, in many cases the preferences file also stores the e-mail (POP) and FTP (publish) passwords.

The implication of this hole is that a JavaScript enabled page can open the preferences file and upload all information contained within it to a remote web server. This can be exploited to capture visitors' e-mail addresses and to gather information about the user's network configuration. The worst risk is that the user's e-mail password will be disclosed. Since the e-mail password is, in many cases, the same as the user's LAN login password, this exposes organizations to a potential route of attack.

All versions of Netscape Communicator through 4.04 are vulnerable. Netscape version 4.05 reportedly fixes the problem.

French software consultant, Fernand Portela, uncovered this bug.

### **2.5.5 Interception of Files on User's Local Machine (October 16, 1997)**

Microsoft Internet Explorer 4.0 for Windows 95/NT is vulnerable to pages that allow the remote Web site operator to spy on the contents of any text, image or HTML file located on your machine, or any file located on a mounted file server. Firewalls cannot protect against this attack, and the browser is vulnerable even when running in "High Security" mode. The Macintosh versions of IE 4.0 are apparently not affected.

The bug, discovered by German consultant Ralf Hueskes and known as the "Freiburg attack," involves the trick of using JavaScript to create an invisible frame 1x1 pixel wide. While the unsuspecting user browses the remote site, a JavaScript program running in the invisible frame scans the user's local machine and file shares for files with well-known names, and may then upload them to any site on the Internet. The bug does not allow JavaScript to modify or damage files. Microsoft had issued a patch for this bug shortly.

### **2.5.6 Ability to Monitor the User's Session (August 29, 1997)**

A group of related bugs allows JavaScript pages to monitor all pages the user visits during a session, capture the URLs of documents viewed, and transmit the information to a host somewhere on the Internet. Some variants of the bug allow the malicious page to capture the contents of fill-out forms, cookies, and information about other elements on the page. Information can be stolen even if the user is viewing "secure" pages encrypted with SSL, and users working behind corporate firewall systems are as vulnerable as those connected directly to the Internet. The only risk is to the user's privacy, however. Data and software located on the user's machine cannot be modified.

Most variants of the bug involve the ability of JavaScript pages to open up an invisible window. Unable to see the window, the user would not be aware that a JavaScript program continues to run even after leaving the page that launched the script. Other variants of the bug open up a new browser window and entice the user into using the window for subsequent browsing.

Despite many attempts to quash this group of bugs, variants reappear at almost monthly intervals and go under such cute names as "the Bell labs bug," the "Singapore bug" and the "Santa Barbara bug." There have been so many patches and releases, it is difficult to keep track. However, the following browsers are known to be vulnerable:

- *Microsoft Internet Explorer 3.01, Windows 95/NT*
- *Microsoft Internet Explorer 3.02, Windows 95/NT*

- *Microsoft Internet Explorer 4.0 Platform Preview, Windows 95/NT*
- *Netscape Navigator 3.0, 3.01, and 3.02, all platforms*
- *Netscape Communicator 4.0 and 4.01, all platforms*
- *Netscape Communicator 4.02, Windows 95/NT*

### **2.5.7 Information Leakage across Frames (July 10, 1997)**

A related type of bug involves the leakage of information across browser frames. To understand why this is a problem, consider two different sites sharing the same browser window, one in each frame. It would be obviously undesirable if a JavaScript program downloaded from an untrusted site could "spy" on the contents of a frame from another site, particularly if that frame contained confidential information.

JavaScript closes some of the holes but not others. A JavaScript could not recover the URL of a document downloaded from another site, but it appears that it can steal a listing of the following document elements:

- *URLs of all in-line images*
- *Other in-line image information, such as width and height*
- *URLs of all applets*
- *URLs of all ActiveX controls*

This means that if a JavaScript page can trick the user into leaving a frame open, it can silently monitor all activities, recording the URLs of all in-line images in documents. Although it could not recover the properties e.g. URL of the document itself, this hardly matters. The vast majority of images on the Web are local.

Although it only captures inline image information from a single document, be aware that it could catch the entire image URLs that are viewed and upload them to a server somewhere.

This bug affects Netscape Navigator 3.0, 3.01 and Netscape Communicator 4.01. It is fixed in 4.02 and 3.03. It does not affect Navigator 2.X or Internet Explorer.

### **2.5.8 File Upload Hole (June 25, 1997)**

Although not strictly related to JavaScript, a bug in the way that Netscape Navigator handles fill-out forms allows JavaScript programs to trick the browser into uploading any local file on the user's hard disk. The user will have no knowledge that the upload has taken place unless he has checked the "Warn before Submitting a Form Insecurely" option in the Security Options dialog box. Even with this option selected, Netscape will fail to produce a warning if the remote server happens to use SSL to establish a "secure" connection.

In order to exploit this bug, the remote server must know the name of the local file in advance. However, this is still a big problem. Large amounts of sensitive information, including system passwords, are stored in files with known names.

Netscape Navigator 2.0, 3.0, 3.01, and Netscape Communicator 4.0 are all affected by this bug. Netscape Communicator 4.01, released June 21, contains a patch for this problem. Version 3.02 of Netscape Navigator is also expected to correct the problem.

This will not be the end of the story. As hackers around the world are uncovering more security loopholes, this will keep all programmers and security experts busying as long as the Web lasts.

## **2.6 Netscape Security Policy**

Netscape has designed a security policy for JavaScript executing in the web browser. The policy is known as Same Origin Policy. Though without formal documentation, we have discovered that the Internet Explorer has adopted a similar policy too.

The same origin policy works as follows: when loading a document from one origin, a script loaded from a different origin cannot get or set specific properties of specific browser and HTML objects in a window or frame.

This policy prevents JavaScript from a site to read into the information of the other sites. Hence, would not be able keep track of what are the URL sites that the user has visited nor alter the information displayed to the user. One could liken this security policy as a "security sandbox" as to Java applet.

## 2.7 Introduction to Script Signing

As a Java Applet could be signed, so do the JavaScript. Signed script requests expanded privileges, gaining access to restricted information. However, only the Netscape Object Signing tool allows script signing, which means signed script is only supported by Netscape browsers.

Object Signing uses techniques based on the X.509 v3 certificate standard and several Public Key Cryptography Standard (PKCS) specifications published by RSA Data Security, including the PKCS #7 signing and encryption standard. Among other things defined by RSA, Object Signing only interested in:

- *create a digital signature for any data*
- *use the digital signature and the appropriate certificates, or digital IDs, to identify the entity that created the signature*
- *use the digital signature to ensure integrity of the signed data*

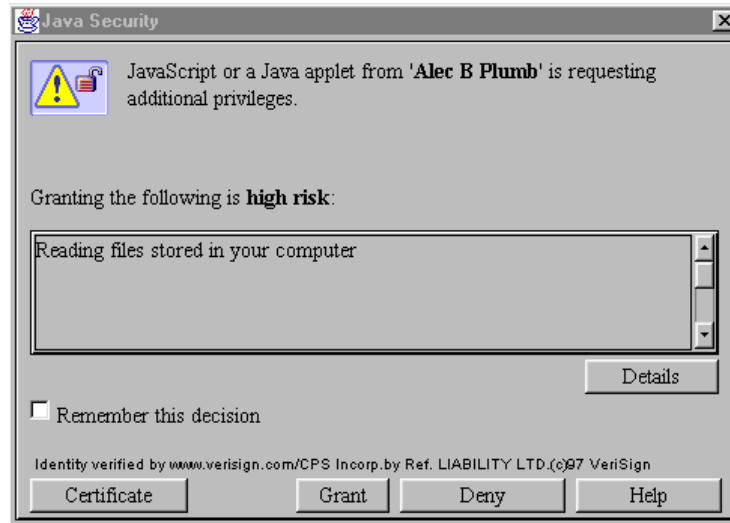
On the Internet and on some intranets, reliable identification of the signers is possible because of the existence of independent, widely recognised certificate authorities (CAs), such as VeriSign, that can authenticate the identities of individuals, companies, or other entities and issue electronic identification documents called certificates or digital IDs. A certificate identifies some entity - much like a driver's license or an employee ID card. CAs can also be internal corporate authorities who authenticate identities and issue certificates for use within an intranet or extranet.

The ability to associate a digital signature with a particular entity allows users and network administrators to decide which sources of software they want to trust and to identify software signed by those entities.

Apart from identifying the signer, Object Signing is used to detect any tampering and to check the integrity of the signed script. Making even a minor change to the script, such as adding a comma in a text file, results in a different, thus invalid digital signature.

## 2.8 Granting of Privileges

However, it is important to note that signing a script could only provide the originality of the script and it is up to the user whether to acknowledge the signer's digital certificate and accept it fully. When the browser detects a signed script in the HTML (i.e. by keyword ARCHIVE), it would pop up a dialog box requesting for extra privileges as shown below.



From the above dialog box, user could check the digital certificate and decide whether to grant or deny the request.

## 2.9 Conclusions

As we could see from the earlier section of “Known Security Holes of JavaScript”, most hackers are exploiting on the browser loopholes, rather than the script itself. No software companies could ever come up with a perfect browser with the perfect security mechanism. To be complete safe, one could completely disable the scripting capabilities from the browser settings no matter how tempting or exciting the provider has promised. But how many users would have such good self-discipline?

## 3 Plug-ins

### 3.1 Overview of Plugin Security

Netscape have offered a method of extending the capabilities of their browser through the addition of native executables, "plugins". Navigator plug-ins have to be downloaded and installed. If you go to a web page that contains a file requiring a plug-in that you do not have, you will usually receive a message asking if you want to get the plug-in. They are not very different from ActiveX controls, except that the method of installation is not automated, so users have to manually find, download, and install the plug-ins. In practice the set of plug-ins is usually restricted to a few well-defined media types, invariably available from the website of the originator of the media format. For example, Adobe Acrobat, Apple QuickTime, Macromedia Shockwave, and Microsoft PowerPoint all have plug-in viewers available.

Microsoft designed ActiveX so that controls will install automatically when you get to a web page that contains a file needing an ActiveX control. For controls set up this way, you may get a warning message telling you that a control is about to be installed. Many multimedia controls still need to be obtained from the developer, but are installed automatically. Shockwave is a good example of this. All you need to do is go to the Macromedia site with Explorer and click on the link to install the ActiveX control. The rest happens automatically. The next time you go to a "Shocked" website, the Shockwave control loads and plays the movie.

### 3.2 Known Shockwave Plugin vulnerability

A good example of how plug-ins can cause security breaches is the Macromedia Shockwave plug-in. The original problem surfaced in 1997 where three potential security holes through Shockwave were discovered that enabled information in your e-mail, on your hard drive or on your corporate intranet to be copied to a distant server.

Any information that a Shockwave movie has access to can potentially be copied to a website on the Internet. This information could then be sent back to a server CGI program that accepts this type of information using long URLs, where the information is passed as a query within the location through the *getnettext* command. This can all happen without the user even noticing.

Users of Corporate intranets who have Shockwave installed are potentially vulnerable. A malicious Shockwave movie could have access to your internal website and all other websites that you can access and then send the confidential information to a designated website on the Internet, as long as the Shockwave movie is running behind the firewall and providing you know the URL. Thus, users' web servers are not fully protected by company firewalls against hacker attacks. Users of Microsoft IE should be aware that Shockwave would automatically install when entering a page with a Shockwave movie on it. They are presented with an Authenticode digital certificate, however the signature on it is from Macromedia and not the author of the potentially malicious Shockwave movie.

A developer can use Shockwave to access the user's Netscape email folders. A malicious user can read and copy someone's private e-mail, including supposedly deleted ones, without their knowledge. This is done by assuming or guessing the name and path to the mailbox on the users hard drive. For example, names such Inbox, Outbox, Sent, and Trash are all default names for mail folders. For windows this could be:

```
mailbox:C:/Program Files/Netscape/Navigator/Mail/Inbox?number=0
```

The developer uses the Shockwave getnettext command to call Navigator to query the e-mail folder for an e-mail message. The results of this call can then be fed into a variable, and later processed and sent to a server.

There are still some problems with Shockwave. On 15th March 1999, there was another security alert. Part of Shockwave's automatic update feature sends Macromedia the URLs for the Websites users have visited, these are collected to determine the most popular sites using Shockwave animation, they then assist those sites in making their animations smaller and faster with the aim of exerting some quality control on Shockwave implementations.

However, Macromedia found itself receiving hundreds of Shockwave users' user names, passwords, and other information that was included in the URLs to some password-protected sites. An update has been posted to the Shockwave site; version 7r205, which has started combing through the incoming URLs to strip out that information.

### **3.3 Conclusions**

Plugins have no security mechanisms built in and it is perhaps surprising that plugins that are more hostile do not exist. Users have almost no resistance to downloading and installing a plugin and it would be a simple matter to create a hostile plugin and create a web page that promised some attractive capability if the plugin was loaded and installed. However, manufacturers seem to be currently relying on the publicising of hostile plugins and hoping that this measure is enough. We are not convinced.

## 4 Java

### 4.1 History

The background of Java is in the C and C++ programming languages. C was an imperative language designed in the early 1970's and was the language in which the Unix operating system was written. By the early 80's, it was the most popular language among software developers and researchers, and at the same time, object-oriented programming was gaining influence. C++ was the result: it contains all the features of the C language with the features of object-oriented programming added on. This language has in its turn become very popular, but it is a large and therefore difficult language to learn and it was felt that a new, more radical approach was required.

In 1990-91 the Java language was designed by a team at Sun Microsystems led by James Gosling (although the language was at the time called Oak). The language was first intended for use in "information appliances" such as cellular phones. By 1993, it became clear to Sun that the new language, now called Java, could be used to provide animation and interaction on the World Wide Web, which was rapidly becoming popular at that time. Java caught the popular imagination in 1995 with the announcement that the next version of the popular Netscape Navigator browser would be able to download and run Java applets.

There were two issues which needed to be dealt with before applets could be routinely downloaded from the Web and run on a local computer. The first is security and relates to the dangers of downloading a program from a remote site and running it on our own machine, which could result in computer viruses destroying our hard disk or copying our files back to the originating Web site. The Java language has been designed to cope with this threat. As we shall see, an applet which is downloaded from a remote site by a Web browser, is very restricted in what it can do.

The second issue was platform independence. The local computer might be a PC, a Macintosh, or a Unix system and each has a different machine code. The solution arrived at by the Java developers was to compile the Java source code not into some particular machine code, but into a machine code (called byte code) for some virtual machine, and this is what is downloaded to the local machine.

We must therefore have a program on our local machine which is capable of interpreting the byte code into the appropriate machine code for local execution. The program is known as the Java virtual machine (VM). Most browsers nowadays have this facility: they are Java-aware; they contain an interpreter, so when they read an HTML page containing a reference to a Java applet, they can download the byte codes and execute the applet while they are displaying the page.

Java attracted attention of programmers throughout the world when it was first released. The developers were attracted to Java for different reasons: its cross-platform capabilities; its ease of programming; its robustness and memory management; its security; as well as other reasons. The nature of the Internet created a new and largely unprecedented requirement for Java applets to be free of viruses and Trojan Horses. Hence, the early work on Java focussed on just that issue: Java applets were considered safe because the code itself cannot perform any action that is harmful to the users computing environment. While some of Java's security features are automatically part of all Java programs, many of them are not.

We are concerned with Java's applet-based security model – the security model which is embodied by Java-enabled browsers. This model is designed for the Internet and is somewhat restrictive and is not the most appropriate for most other programming situations. It should be stressed that the security model of any particular browser is ultimately up to the provider of the browser. Some browsers allow us to change the security policy the browser uses, but many do not.

## 4.2 Java Language Security

The first aspects of Java security that we will consider are those which are built into the Java language itself. Within a Java program, every entity that is every object reference and every primitive data type has an access level associated with it. The levels are private, default, protected and public. Private means that an entity can only be accessed by code from within the class in which it is defined. Default means access is only possible from within the package of classes which includes the class in which it is defined. Protected is similar to default but includes the possibility of access from sub-classes of the defining class. Public means access is possible from within any class. These are the levels referred to in the first of the Java rules given below.

- *Access levels are strictly adhered to in Java;*
- *Programs cannot access arbitrary memory locations: Java does not use pointers*

- *Final variables are immutable. A subclass cannot override a final method;*
- *Variables may not be used before they are initialised;*
- *Array bounds must be checked on all array accesses;*
- *Objects cannot arbitrarily be cast into other objects.*

The Java compiler is the first opportunity to enforce Java's language rules. It is responsible for enforcing all the rules except the last two, which cannot be enforced until runtime.

### **4.3 The Sandbox Concept**

The idea of a sandbox lies behind Java's security model. When you allow a program to run on your computer, you want to confine the program's 'play area' within certain bounds. You may allow it access to certain resources ('toys'), but in general you want to confine it to its sandbox. The Java sandbox is responsible for protecting a number of resources, and it does so at a number of levels. The user's machine has access to many things: local memory; local file system and other machines on the local network; web server, which may be on the local server or on the Internet; data flows through this entire model, from the user's machine through the network.

A number of different-sized sandboxes can be envisaged and the one often thought of as the default state for the sandbox is the one in which the program has access to the CPU and its own memory as well as access to the web server from which it was loaded. The sandbox can be expanded or shrunk depending on the trust the user has in the program. In practice, this falls back on how much trust the author of the browser puts in applets downloaded from the Internet, and it is the browser application which is responsible for establishing the parameters of the Java sandbox.

### **4.4 The Components of the Sandbox**

The default sandbox has three related components:

- *byte code verifier*
- *class loader*
- *security manager*

Between them, the three parts perform load time and run time checks in order to restrict file system and network access.

#### 4.4.1 Byte Code Verifier

The byte code verifier is an internal part of the Java virtual machine: it helps enforce memory protections for all Java programs. In the case of applets, whose byte codes may have come from an unknown source, how can we know that they are legal? The byte code verifier automatically examines byte codes of classes as they are loaded into the Java virtual machine. It is the next link in the chain of responsibility of enforcing the rules of the language. The byte code verifier can prove the following:

- *The class file has the correct format;*
- *Final classes are not sub-classed;*
- *Every class has a single super-class;*
- *There is no illegal conversion of primitive data types (e.g int to Object);*
- *No illegal data conversion of objects occurs;*
- *There are no operand stack overflows and underflow.*

The byte code verifier not only helps prevent malicious attacks from violating the rules of the language, it can also detect simple programmer errors. However, it is not used on all classes. Like many security-related features in Java, byte code verification only applies to certain classes. In Java 1.1 and earlier, local classes are deemed trusted and are not subject to byte code verification, whereas classes loaded from another location (e.g. a file- or HTTP-based URL) are not deemed trusted and must be verified. In Java 1.2, this policy has changed and all classes except those in the core Java API are verified. Browsers always ensure that the code imported to run an applet is verified.

Like the compiler, the byte code verifier cannot completely guarantee that the byte codes follow all the rules of the Java language: it can only ensure that the first four are followed. The remaining security protections of Java must be enforced at runtime by the virtual machine.

#### **4.4.2 The Class Loader**

All Java objects belong to classes. Class loaders determine when and how classes can be added to a running Java environment. They provide memory management: part of their job is to ensure the vital parts of the run time environment are not corrupted or replaced

Class loaders perform two functions. First, when the VM needs to load the byte code for a particular class, it asks the class loader to find the byte code. Each class loader can use its own method for finding requested byte code files: it can load them from a local disk, fetch them from across the network, or it can create the byte code on the spot. Second, class loaders define the name spaces seen by different classes and how these name spaces relate to each other. It is important to keep internal classes separate from external classes, and only the class loaders maintain the necessary information about the class files to achieve this.

#### **4.4.3 The Security Manager**

The Java security manager is responsible for determining most of the parameters of the Java sandbox; that is it determines whether particular operations should be permitted or rejected. The security manager is a single object that performs run time checks on dangerous methods. It is consulted whenever a potentially dangerous operation is attempted. It takes into account the origin of the requesting classes when making a decision and can if necessary veto the operation.

When the security manager rejects an operation, it throws a security exception. Typically, the exception propagates up through all the methods in the thread that made the call; eventually the top-most method receives the exception, which causes the thread to exit and a message is displayed in the Java console. If the security exception is not thrown because the operation is not rejected, then the method in the security manager just returns and processing proceeds as expected.

Each VM can only have one security manager installed at a time and once it has been installed, it cannot be un-installed (except by restarting the VM). Java-enabled browsers install a security manager as part of other initialisation before any potentially untrusted code has a chance to run.

## 4.5 Different Classes of Security: Towards Java 2

The previously clear distinction between applications, which were assumed trusted, and applets, which were totally untrusted, has gradually faded. The Java security package is a set of classes that were added to Java 1.1 (and expanded in 1.2) which allow a class loaded from the network to become trusted if it is digitally signed by someone the user has decided to trust. The idea is to allow such trusted code the same level of privilege as is usually afforded to built-in code.

With Java 2, things get even more complicated: the notion of partial trust is introduced, which when implemented by the VM allows for finely grained security policies to be set up and implemented by the language. Effectively, code that is partially trusted can be placed in a specially constructed custom sandbox; this is implemented by the three components of the VM discussed above together with access control. For example, an applet designed for use in an intranet setting could be allowed to read and write to a company's database as long as it was signed by the system administrator.

Clearly such controlled relaxation of the constraints of the restrictive default sandbox for all mobile code could be very useful for developers. Introducing permissions granted according to trust level opens up many new application areas, including things like spreadsheet applets, games with stored high scores, Web sites that recall your preferences and many more.

There are many choices to be made with the new finely grained security models. They cannot be left wide open for the user who typically has neither interest nor expertise in security policies, whilst system administrators may find them an unwelcome burden. It is likely that the good browsers will choose to offer users packaged sets of capabilities for granting on the basis of trust rather than offering a long list of privileges to be chosen for each program according to its signature.

The use of digital signatures requires an adequate PKI and confidence in Certification Authorities: the problems associated with these may well be a limiting factor for the early success of Java2's brave new approach.

## 4.6 Java applet vulnerabilities

Java applets can be classified as either:

- *Malicious – range from annoying to crashing the browser – found on the Internet*
- *Attack – completely take control of a machine – not found on the Internet*

#### 4.6.1 Malicious Java Applets

Malicious applets can be encountered on the web and so are definitely in the wild.

A malicious applets purpose may be:

- *Denial-of-service*
- *Loss of privacy*
- *Annoyance*

The actual effect of a malicious applet varies widely. A simple malicious applet might play an annoying noise continuously when a web page is loaded into the browser. This is simply frustrating for the user and does not affect the running of the web browser, the computer, or any of the data. However, it is possible for the applet to continue playing the noise after the users has moved to another web page and to continue until the web browser is stopped, even if Java is subsequently turned off. The applet is behaving entirely consistently with the Java security model and there is no security mechanism to prevent this simple type of malicious applet.

A more sophisticated malicious applet might try to prevent applets from named sites from executing. For example, an applet could begin to run when a web page is visited. The applet would remain running in the background and if a particular web page were visited, then the applets on that page would be terminated by the malicious applet running in the background. This behaviour was permitted in the Java 1.1 Security Manager when an applet was allowed to manage the running of applets from web pages other than its own. This particular fault was fixed in the Java 2 Security Manager.

A malicious applet whose intention was a denial-of-service, could crash the browser. There are many ways to initiate a denial-of-service attack, but a simple way is to start an applet that performs some innocuous action in the foreground. A background action is started that sleeps for some time, then begins a very intensive operation, that effectively uses all of the CPU power available and disables the browser. For more effect, the malicious applet could also progressively use all of the memory, which may also cause other applications to crash.

The effect of these denial-of-service attacks are inconvenient for users who are surfing the web recreationally but could have significant effects on a user who is trading online. There is no provision in the Java security model to combat any of these attacks. A more serious denial-of-service attack is to create large windows on the screen that cover both the browser and other applications. The creation of windows would continue indefinitely until the browser is killed. However, an applet could deny access to the keyboard and the mouse as the windows are created at such a fast rate that events generated from the keyboard or the mouse may be ignored.

Forged emails are common on the Internet, but can usually be traced through the IP address that will be contained in the 'Received:' field. However, a malicious Java applet can create forged emails from within a users computer and it will be impossible to recognize the email as forged. The applet will operate entirely within the Java sandbox and there are no mechanisms in the Java security model to control this behaviour. For examples of further malicious apps, [www.rstcorp.com/javasecurity/links.html#applets](http://www.rstcorp.com/javasecurity/links.html#applets) is a good web site.

As noted, the Java security model does not have any controls to stop the malicious applet attacks, except the ability to only execute digitally signed applets from a trusted source. We address the current shortcomings in this defence later in the report.

#### **4.6.2 Attack Applets**

Attack applets are not seen in the wild and only exist in the laboratories of the Java security researchers. There are no known instances of an attack applet on the Internet. Why not? Luck, probably. The security researchers are interested in making the good Java security model better, while the bad guys are more interested in softer targets such as mis-configured web server applications and host operating systems.

The purpose of an attack applet is:

- *Anything*

An attack applet can take complete control of your machine. It has access to local disks, network connections, all applications, and all data. It can rewrite data, destroy data, or use the machine as a stepping off point into the local network.

Many of the Java security problems that enable attack applets are the result of implementation errors in the Java virtual machine, but some are the result of incorrect specification and have resulted in the design of Java changing. Approximately 20 serious security problems have been found in Java since February 1996.

Exact details of Java attack applets are difficult to obtain, especially those where the weaknesses still exist. The security researchers are pleased to have found the problems, but are not keen on other people reproducing the attack applets until the weaknesses have been addressed. We have included a detailed explanation of a single type of attack applet that relied on an implementation weakness that has now been addressed.

#### **4.7 Attack Applet using Type Confusion**

A type confusion attack confuses the Java system about the types of data objects that are being manipulated. The Java Security Manager uses the type of a data object to control what actions that object can perform. When the type of an object is confused, it is possible for that object to perform actions that are not normally available to it.

For instance in version 3.0beta5 of Netscape, a Java applet could declare itself to be actually an array of Java applets. An error was generated, but the Java virtual machine would still believe that the applet was in fact an array of applets. Subsequently the Java virtual machine would allow access not only to the first applet which really existed, but also to the subsequent non-existent applets in the array. Although the subsequent applets did not exist, in their place were other Java objects. In this way, an attack applet with a little experimentation could gain access to any other object in the Java system. The result was a full penetration of the system with the ability to read or write any file or control any part of the system.

Type confusion has formed the basis of many attack applets and the Princeton University team that has pioneered much of the security research surrounding attacks on Java, has built a type-confusion toolkit enabling the simple type-confusion security breach to be turned into complete system penetrations. This toolkit has obviously not been released but has been revised to work with Java 2.

## **5 ActiveX**

### **5.1 Overview**

Active Content refers to executable programs downloaded into the network and ultimately into the local workstation while Web browsing. These downloads may occur with, or sometimes even without the knowledge of the user. Active Content includes all codes designed to enhance the Internet browsing experience by making it more interactive such as Microsoft's ActiveX components. Although most of these executables are innocent and harmless, they can be used with malicious intent, to attack networks and systems.

Since ActiveX controls are native programs, they can do all the things native Windows programs can do. For example, they can read and write files, perform network administration, and determine the capabilities of the system they are running on.

ActiveX represents a significant security risk, because such applications have the potential to impact a user's local machine. An ActiveX application, for example, can fire up a desktop application using specific commands. While this offers the possibility for combining desktop capabilities with the Web, it also provides an indication of how vulnerable systems can be to malicious applications and hackers.

### **5.2 Introduction**

Developed by Microsoft corporation, ActiveX is a technology that has become an essential part of Microsoft applications and tools; they are even finding their way into Microsoft operating systems.

A common advantage of Microsoft is its ability to share information between applications. This type of technology was known as OLE (object linking and embedding), this technology can be seen today in widely used OS like Win 3.1, Win98, etc. OLE was a technology for creating compound documents. Which mean that it allowed you to do things such as 'cut and paste', which proved to be, was very useful.

The next release of OLE, (which was not focused solely on compound documents), named OLE2 introduced Microsoft's Component Object Model COM. COM grew out of the OLE architects' desire to provide a more general mechanism for allowing one piece of software to provide services to another. Very quickly, then, COM began to be used in technologies that had nothing whatsoever to do with compound documents. So now Microsoft had a nice, general infrastructure technology where in the spring of 1996 named ActiveX.

COM is really the heart of ActiveX and other things dealing with Windows. For example, if you have ever used Word 97 and have ever inserted a spreadsheet you can see that although only Word is open you can still work with an Excel spreadsheet normally. This is known as embedding; it is when one type of document or application is placed inside another document or application. The original OLE did not have this capability.

ActiveX components are written as DLLs or just small applications that are made in some other language like Visual Basic or C++, and are meant to run (loaded) in another environment called a 'container' since they cannot run on their own. You do not have to install ActiveX because Windows already utilise such things as COM and OLE (as mentioned above). However, in order to use ActiveX you must have a 'container' that supports ActiveX. For example, web browser like Microsoft's Internet Explorer and Microsoft Word are ActiveX containers.

ActiveX components perform almost the same functionality as Java Applets allowing much more to be done with your web page such as the addition of interactivity. These interactive controls to Web pages can be anything from a single push-button to a complete spreadsheet. However, Java Applets can only run in an Internet browser environment where ActiveX will run in any Windows container, which can be a Visual Basic Form, Internet Explorer, etc. By this, ActiveX can be used in places other than your web browser. This makes it much more versatile and robust.

Unlike Java applets, however, ActiveX controls have full access to the Windows operating system. This gives them much more power than Java applets, but with this power comes a certain risk that the applet may damage software or data on your machine. To control this risk, Microsoft developed a registration system so that browsers can identify and authenticate an ActiveX control before downloading it. Another difference between Java applets and ActiveX controls is that Java applets can be written to run on all platforms, whereas ActiveX controls are currently limited to Windows environments.

### 5.3 ActiveX security

ActiveX, on the other hand, places no restrictions on what a control can do. Instead, each ActiveX control can be digitally "signed" by its author in such a way that the signature cannot be altered or repudiated using a system called "Authenticode." A trusted "certifying authority", such as VeriSign, then certifies the digital signatures. When a digital certificate is granted, the software developer pledges that the software is free from viruses and other malicious components. If you download a signed ActiveX control and it crashes your machine, you will at least know whom to blame.

This security model places the responsibility for the computer system's security on the users. Before the browser downloads an ActiveX control that has not been signed at all, or that has been signed but certified by an unknown certifying authority, the browser presents a dialog box warning the user that this action may not be safe. The user can elect to abort the transfer, or may continue the transfer and take his chances.

The ActiveX certification process ensures that ActiveX controls cannot be distributed anonymously and that third parties cannot tamper with a control after its publication. However, the certification process does not ensure that a control will be well behaved. Although it is unlikely that signed and certified ActiveX controls will behave in a malicious fashion, it is not impossible. However, naive users who have changed Internet Explorer's restrictions on active content to "Low Security", or who agree to download and execute the controls despite the warnings, are vulnerable to attack by this means.

One other problem with the ActiveX security model is that it is difficult to track down a control that has taken some subtle action. For instance, an applet could silently transmit confidential configuration information from the user's computer to a server on the Internet, seed the LAN with a virus, or even patch Internet Explorer so that the code authentication engine no longer functions correctly. This type of action may escape detection completely or at least for a long period. Even if the damage is detected immediately, Internet Explorer offers no secure audit trail that record which ActiveX controls were downloaded. This makes identifying the control responsible for damaging your system a non-trivial task.

Currently Microsoft's Internet Explorer provides two ways of protecting one's data and machine from malicious use of ActiveX.

- *Blocking / Allowing of all ActiveX:*  
*This simple on/off setting either exposes the network to a high security risk (on) or denies users the business benefits of useful ActiveX code (off).*

- Allowing of ActiveX, signed by trusted entities:  
*The digital signature approach demands that each client browser be configured with the proper security policy, including a current list of trusted signatures. This approach relies on every user to enforce the network's security policy.*

From an organisation's perspective, to retain centralised control over the organisational security, the blocking of ActiveX or validating of digital signatures should be performed automatically at the gateway level, and according to the organisation's downloadable security policy, rather than leaving these crucial safety issues in the hands of the users.

The most dangerous situation, though, is when the program is signed by someone you do not know anything about. You would really like to see what this program does, but if you reject it, you will not be able to see anything. So you rationalise: the odds that this particular program is hostile are very small, so why not go ahead and accept it? After all, you accepted three programs yesterday and nothing went wrong. It is just human nature to accept the program.

Even if the risk of accepting one program is low, the risk adds up when you repeatedly accept programs. In addition, when you do get the one bad program, there is no limit on how much damage it can do. All actions requested by a Web page need to be considered as hostile, until proven safe. Similarly, all executables on a machine should be considered hostile by the browser until they are proven safe. The equivalent of a Java sandbox is needed for all actions requested by ActiveX

## **6 Protection against hostile applets**

There are several approaches to defending against hostile applets and most of these approaches are applicable to all of the technologies examined in this report. There are software solutions on the market which can help prevent attacks by hostile applets, but this technology is still in its early stages. We have surveyed products available, however we have not performed any evaluations, so product information in this section is taken from marketing material or product reviews.

### **6.1 Turn it off**

A simple question to ask is “Can you do without it?”. Disabling Active Content is still possible in web browsers and email clients. This option is the only way to guarantee complete security while still allowing web browsing and emailing to continue unimpeded. However as some of the technologies underlying Active Content start to be used pervasively in operating systems, how soon will it be before turning of Active Content deprives users of pieces of their operating system?

### **6.2 Use latest maintenance release of browser version**

Microsoft and Netscape, the publishers of the major browsers address many security vulnerabilities quickly. However, a large corporation may have upwards of 50,000 copies of a browser deployed and it may not be possible to roll out a new version of the browser instantly or even quickly. There may need to be a period of testing with internal applications and planning for the deployment of the new version.

The inability of corporations to roll out new software versions quickly provides a window of opportunity for hostile applets. It will require new models of software distribution to enable these windows of opportunity to be minimised and there seems to be no software distribution model currently available that can be used easily to close this window of opportunity across the thousands of desktops and laptops in a typical large corporation.

### **6.3 Filter at the client for known malicious applets**

One approach to limiting the opportunities for hostile applets is to build lists of applets that can be denied access to a machine. There are two problems with this approach. Firstly, it is simple for an application to change its name. The solution to this is to include a signature in the black list enabling a pattern of bytes in the applet to be identified. However, the second problem is that unfortunately it is trivial for an applet to mutate its own byte code every time it is downloaded.

So by definition, black lists of applets are easily circumvented. However, they should not be disregarded as they do stop well-known hostile applets from easily multiplying on the internet.

McAfee Associates' WebScan relies on databases of known viruses and hostile applets to block harmful code from accessing a client PC. Trend Micro offers WebProtect, which screens for viruses as well as known hostile applets. WebProtect also provides configurable blocking options, allowing administrators to name trusted partners from whom they will accept signed Java applets and Active X objects.

SurfinShield Corporate published by Finjan is a PC-based hostile applet defence. SurfinShield incorporates Auto-Immunity, a centralized database of hostile applet names that is updated in real-time and shared by all SurfinShield clients. Hostile applets named in this database can be denied access to users PCs. SurfinShield Corporate can also automatically kill any Java or ActiveX applet that tries to breach the security rules before the code is allowed to do damage.

Centralised reporting of specific mobile code violations is supported and information about violations is distributed to other PCs running SurfinShield in the same organisation. So, if one desktop faces a mobile code attack and learns from it, all other SurfinShield Corporate desktops will be updated to automatically kill the same hostile element. The product includes Finjan's X-Box de-militarised zone (DMZ), which runs ActiveX applets in a separate memory space from the web browser, similar to the Java sandbox.

Additionally the Auto-Launch Blocking feature monitors a user's Web browser to prevent the automatic launching of any MS Office application without user knowledge. SurfInShield also provides enhanced security to the Java sandbox by extending security options for the Java Security Manager. A monitor bar allows users to view and control each Java applet and ActiveX control that are running on their machine. Finally, URL filtering using URL paths can be registered to create "block and allow" lists that give security administrators enhanced content-filtering capabilities.

An alternative is eSafe Protect that protects PCs from viruses, as well as hostile Java Applets, ActiveX controls, plug-ins and scripting languages. eSafe Protect does not block the entry of unknown code, Java applets, or ActiveX controls into your computer, but instead all your mobile code is run in a protected sandbox environment, isolated from the rest of the PC. In this closed system, the behaviour of every object is closely monitored and applets that perform hostile actions are terminated.

#### **6.4 Filter at the firewall for known malicious apps**

An alternative for an entire network is to filter malicious applets at the network firewall. This approach suffers from the same shortcomings as scanning for hostile applets at the client.

eSafe Protect Gateway is an example of such a product that looks for hostile applet names passing in traffic in the firewall. It scans and cleans FTP, SMTP, and HTTP traffic and remembers sites with viruses and hostile applets to block them completely in the future. It also can optionally strip all macros, cookies, and active content of certain types and block email or web pages based on keywords or URLs.

An alternative is Finjan SurfInGate, which attempts to provide companies with centralized protection against hostile Java applets only. However, by focusing on Java, the product fails to protect against ActiveX controls, browser plug-ins and Web browser security holes. Java applet security is configured centrally for an entire network and then different rights can be created for groups of users or individuals.

SurfinGate can be configured to block all Java applets or just those with one or all of the following characteristics: illegal file access, illegal network access, or loading of classes dynamically. However, the third option can cause problems, as many useful applets contain this characteristic, which essentially launches another applet. The product maintains a library of known Java applets and assigns access rights by applet allowing known hostile applets to be blocked while letting beneficial applets pass. There is also an option to configure the applet as either allowed, blocked, allowed with certificate, allowed with access control, or allowed with access control and certificate. The 'allowed with access control' option makes it easy for applets from an IP address within the company intranet to be given free reign.

McAfee's WebShield X also scans for known hostile applet names while also featuring a content scanner that lets administrators filter outgoing messages that might contain confidential information

## **6.5 Route all mobile code through sacrificial machine "CAGE":**

Digitivity, the offspring of British consulting company APM, took a different tack. The company designed a "CAGE" that sat outside a firewall, on which Java applets (or more specifically, applet proxies sent by a browser) were executed. Mimicking X Windows, the CAGE only transferred the GUI information from the Java applet back to the client's browser. It reduced risk significantly because the applet did not run on the user's machine. However there was a question about how this technique could have scaled to support a large organisation. However, the product was discontinued when in the middle of 1998, Digitivity was bought by Citrix Systems Inc.

## **6.6 Ring around the data:**

Rather than preventing hostile applets from entering a system, some solutions protect specific resources on a PC. For example, CyberMedia's Guard Dog prevents Java applets from accessing any files that a user wants protected. Anytime an applet attempts to access these applications, the product sounds an alarm and the user decides whether or not to let it execute. After applets are downloaded, Guard Dog watches for suspicious activity, deleting of files, reformatting disks, silently dialling out to the Internet, and so on. When any of the suspicious activities occur, an alarm is again sounded alerting the user and allowing them to decide whether to continue.

The downside to this approach is that security decisions are placed back on the user who is not in a position to make authoritative security decisions.

## **6.7 Resource limiting**

One solution to the applets that perform the annoying denial of service attacks is resource limiting, either at the Java virtual machine level or the operating system level. The hostile applet that intends to create an infinite number of windows to crash your browser and perhaps your operating system could be contained by a policy in the Java virtual machine that limited the resources that the applet could consume. A more general solution would be a policy at the operating system level limiting resources that an application could consume.

However, operating systems do not yet contain these resource-limiting mechanisms and newer operating systems on the immediate horizon (Windows 2000 for example) do not contain resource limiting.

## **6.8 Applet signing**

Code signing is an approach that can be applied to all of the Active Content technologies. It does not check for hostile applets, nor does it run the applet in a sandbox. Instead, it is intended to provide users with an audit trail.

A digital signature is attached to an applet providing authentication and integrity. A signature should have a certificate from a third party, proving that the signer is who they say they are. If the third party is a root certificate authority, then if the user trusts the root CA, they can then begin to trust the applet. If the third party is not a root CA, there should be a means to follow a certificate chain back to a root CA or at least to a CA that the user trusts.

If a signed applet causes trouble, the signature on the applet should show who created the applet. It also allows the user to deny access to applets that have a signature that they don't have trust in.

However there are significant problems to relying on applet signing currently:

### **6.8.1 The audit trail is vulnerable**

Audit trails of which applets have been downloaded and which signatures have been checked are vulnerable to changes by the hostile applet. If an applet can remove this audit trail, then it may be able to perform its attack but a user cannot then report the applet as hostile as they cannot identify its source.

### **6.8.2 Damage done may not be visible**

A hostile applet may not cause immediately visible damage. For example, a hostile ActiveX applet may have a correct signature attached but the damage that it does is such that the signature should be revoked so that other users cannot be harmed by the applet. However, if the damage is invisible to the users until some time later, then the signature mechanism provides no protection.

### **6.8.3 The signature checking code is vulnerable**

A hostile applet may attack the code that checks the signatures. If an applet could replace the signature checking component of a web browser with a false component that doesn't correctly check signatures in the future, then that machine is now vulnerable to attacks from any hostile applet, with or without a valid signature.

### **6.8.4 Signed applets may have other vulnerabilities**

A signed applet may be innocuous and perform no hostile actions. However there may be another vulnerability contained in the applet. For example, signed valid applets from legitimate users may contain buffer overflow defects.

### **6.8.5 The signature may have been recently revoked,**

A signature must be checked every time an applet is used and the signature must be checked against an up-to-date list of revoked certificates. If not, it is possible that applets recently discovered to be hostile, will still be allowed to run. If the signature for the applet has been recently revoked, then notice of that revocation must be made immediately available to all machines that need to check signatures. Until a pervasive global PKI exists, signing applets allows a window of opportunity for hostile applets to perform damage on many users machines until revocation catches up with them.

## 7 References

1. "The World Wide Web Security FAQ" by W3C  
<http://www.w3.org/Security/faq/www-security-faq.html>
2. "JavaScript Security" by Netscape Corp.  
<http://developer.netscape.com/docs/manuals/js/client/jsguide/sec.htm>
3. "Netscape Object Signing - Establishing Trust for Downloaded Software" by Netscape Corp.  
<http://developer.netscape.com/docs/manuals/signedobj/trust/index.htm>
4. "Microsoft Windows Script Technologies" by Microsoft Corp.  
<http://msdn.microsoft.com/scripting/>
5. "New Viruses Search For Strong Encryption Keys" by Tech Web  
<http://www.techweb.com/wire/story/TWB19990315S0001>
6. "Java Security FAQ" by Princeton University  
<http://www.cs.princeton.edu/sip/faq/java-faq.php3>
7. "ActiveX and Exploder" by Fred McLain  
<http://www.halcyon.com/mclain/ActiveX/#activex>
8. "ActiveX Security: Under the Microscope" by Web Developer  
[http://www.webdeveloper.com/activex/activex\\_security.html](http://www.webdeveloper.com/activex/activex_security.html)
9. "FAQ About Developing Web Pages Using ActiveX Controls"  
<http://www.mlink.net/~pchou/fuktop/pages/activex.html>
- 10.
- 11.
- 12.
- 13.
- 14.
- 15.
- 16.