A study of the security implications involved with the use of executable World Wide Web content

Christopher Hu
Edith Cowan University

A Study of the Security Implications Involved with the use of Executable World Wide Web Content

Christopher Hu
0920484

For the Award of:
Master of Science (Computer Science)
Edith Cowan University

Supervisor:
Assoc/Prof William Hutchinson

Abstract

Malicious executable code is nothing new. While many consider that the concept of malicious code began in the 1980s when the first PC viruses began to emerge, the concept does in fact date back even earlier. Throughout the history of malicious code, methods of hostile code delivery have mirrored prevailing patterns of code distribution. In the 1980s, file infecting and boot sector viruses were common, mirroring the fact that during this time, executable code was commonly transferred via floppy disks. Since the 1990s email has been a major vector for malicious code attacks. Again, this mirrors the fact that during this period of time email has been a common means of sharing code and documents.

This thesis examines another model of executable code distribution. It considers the security risks involved with the use of executable code embedded or attached to World Wide Web pages. In particular, two technologies are examined. Sun Microsystems' Java Programming Language and Microsoft's ActiveX Control Architecture are both technologies that can be used to connect executable program code to World Wide Web pages. This thesis examines the architectures on which these technologies are based, as well as the security and trust models that they implement. In doing so, this thesis aims to assess the level of risk posed by such technologies and to highlight similar risks that might occur with similar future technologies.

Declaration

I certify that this thesis does not incorporate without acknowledgement any material previous submitted for a degree or diploma in any institution of higher education; and that to the best of my knowledge and belief it does not contain any material previously published or written by another person except where due reference is made in the text.

Signature

Date 2/5/2003

Acknowledgements

I would like to thank Dr Helen Armstrong and Dr Timo Vuori for all of their help in preparing this thesis.

Table of Contents

Table of Figures

1. Introduction and Scope

1.1. Introduction

The idea of malicious program code is as old as modern computers themselves. In 1949, John von Neumann's "A self reproducing program in Theory and Organization of Complicated Automata" (cited in McMullin, 2000) proposed the idea that a computer program could reproduce itself.

When Fred Cohen began researching the idea of programs that replicate by inserting code into other programs in 1983 the idea of the Computer Virus was born. While virus-like code such as "Elk Cloner" (Skrenta, n.d) had earlier appeared on Apple II systems, Fred Cohen's work led to the coining and definition of the term Computer Virus. Cohen defined a virus as "...a program that can 'infect' other programs by modifying them to include a possibly evolved copy of itself" (Cohen, 1984).

In 1988 the Morris Worm (CERT,1997) spread around the Internet with frightening speed. While not the first code of its type, the Morris Worm demonstrated the vulnerability of connected systems to a rapidly spreading attack

The Back Orifice Trojan, released by the Cult of the Dead Cow (cDc) in 1998 (CERT, 1998), received a significant amount of attention. Trojans such as BO2K can grant an attacker almost total control of a victim's machines. While other client-server Trojans such as Netbus provided similar capabilities, Back Orifice still remains one of the most high profile and most dangerous of Trojans. Its current form, known as BO2K is one of the most notable of current Trojans.

Since the 1980s, malicious code has been part of the computer security landscape but this landscape is changing. It is interesting to note that recent years have seen somewhat of a blurring between some of these types of malicious code. In particular the distinction between viruses and worms has narrowed. For example, some articles refer to Melissa as an example of a worm (Sophos Anti-Virus, 2002), others refer to it as a virus (CERT, 1999b) while others refer to it as a hybrid that exhibits the characteristics of both a virus and a worm (Nachenberg, n.d). While CERT refers to Melissa as virus rather than a worm, due to its reliance on human interaction in order to spread, it does acknowledge that the level of human interaction required is minimal (CERT, 1999b). Likewise the Loveletter Worm is also sometimes referred to as a virus (Microsoft, 2002a).

Increasingly email is becoming the major vector for such worms and viruses. However, new technologies such as various forms of executable web content may play an important role in this changing landscape of malicious code. Email has a number of characteristics that makes it an attractive to writers of malicious code as an infection vector. The ubiquitous availability of email allows an attacker potentially affect vast numbers of systems. As a form of personal communication, email allows an attacker opportunities to make use of social engineering techniques to spread malicious code. Finally the lack of intrinsic, integrated security controls means that there are many avenues of attack that can be exploited by the writers of malicious code.

This thesis examines the possibilities for malicious code being implemented using executable web content technologies such as Java and ActiveX. Both Java and Active allow executable code to be embedded within a web page and executed on client machines when that page is viewed. While this can help web developers to create increasingly dynamic and engaging web pages, the fact that untrusted, possibly malicious code is being executed raises a number of security concerns. The architectures of Java and ActiveX will be examined along with the security functionality that they provide.

The addition of executable code to web pages raises several new concerns. This code has a different model of distribution to other forms of software. With this new model of distribution comes a range of new security issues. Such code has the ability to affect confidentiality of information, integrity of data, software and operating systems as well as the availability of systems and services. There are also a number of issues related to the authenticity of such code and the ability of people to deny developing malicious code.

This thesis will argue that while operating system and web application levels security mechanisms are an important layer of defence, executable web content technologies need to implement their own trust and security architectures.

1.2. Scope of Thesis

While the risks facing users of the World Wide Web are many and varied, this thesis is quite specific in its scope. It focuses solely on the risks to World Wide Web users posed by malicious executable web content. In particular it focuses on Sun Microsystems' Java programming language and Microsoft's ActiveX technology. While interpreted forms of executable web content including scripting languages do raise certain security concerns, this thesis limits its scope to binary forms of executable web content.

This thesis discusses the security and trust models employed by Java and ActiveX. This thesis also considers the security mechanisms implemented by the Windows NT/2000/XP line of Microsoft Operating Systems as well as those implemented by common web browser applications.

1.3. Significance of Thesis

While Java, ActiveX and the World Wide Web have now existed for several years, it is important to reflect upon the issues that have been raised by these technologies and to consider those issues that might be raised by the next generations of World Wide Web oriented code delivery mechanisms.

This thesis contends that the World Wide Web and the Internet in general will be one of the major channels for code distribution in the near future. As such it is important to examine the security issues raised by current forms of executable web content so that the next generations of such code can build on this experience.

2. Research Methods and Models

2.1. Overview

This chapter outlines the research methods that will be employed in this thesis.

2.2. Research Questions

This thesis aims to answer several important questions regarding the security risks posed by current web technologies. All of these questions revolve around client machines and consumers of World Wide Web services, as opposed to service and content providers.

Does executable WWW content pose a significant security threat to client machines?

This thesis attempts to determine whether or not there are significant inherent security risks posed by the concept of executable web content. By examining two such forms of executable web content, this thesis attempts to highlight the basic level of risk that technologies such as Java and ActiveX must attempt to guard against.

Do the security mechanisms offered by these technologies provide a suitable level of protection?

This thesis also examines the concepts behind the security mechanisms implemented by both Java and ActiveX. It pays particular attention to the question of whether or not the security models on offer are adequate to offset any inherent security risks (if any) posed by the use of executable web content.

Are there significant differences in the security mechanisms provided by popular WWW browsers?

The security models offered by Java and ActiveX are also examined in the context of the web browsers through which such code will operate. This thesis examines the differences between the security features offered by current web browser applications and assess the role played by such browsers in reducing any risks posed by executable web content technologies. Web browser security features will only be discussed in terms of their relationship to executable web content technologies

Are there significant benefits to be gained from using secure desktop operating systems in conjunction with WWW applications?

Finally, this thesis attempts to determine whether or not there are any real security benefits to be gained from using a desktop operating system that implements various security controls. It examines the code signing, access control and auditing features of the Windows NT/2000/XP line of Microsoft operating systems in order to determine the effectiveness of operating system level controls in guarding against any risks that might be posed by executable web content technologies such as Java and ActiveX.

2.3. Research Validity

This thesis aims to address the research questions outlined in the preceding section. In taking such a qualitative approach, it is intended that this thesis will…

These questions have been chosen in order to examine the security models employed by executable web content technologies, as well as the ways in which these security models interact with the security features offered by certain web browsers and Operating Systems.

By examining the security issues associated with current forms of executable web content, this thesis aims to provide an insight into the types of security issues that will need to be addressed by future generations of mobile code. While this thesis does not seek to define the security architectures that will or should be employed by such generations of code, it does aim to highlight the strengths, weaknesses and limitations of the security models offered by current executable web content technologies.

2.4. Summary

The research questions outlined in this chapter form the basis of this thesis. These questions are addressed after examining the technologies in question.

3. Aims of Security and Threats Posed by Malicious Code

3.1. Overview

This chapter provides an overview of the types of risks posed by various forms of executable code and the aims of computer security that are threatened by these risks. While the security risks posed by malicious code such as viruses, worms and Trojans have been discussed at length in many texts over a number of years, this chapter highlights some unique security concerns raised by the use of executable web content technologies. In particular, it highlights the different models of distribution between traditional stand-alone applications and code delivered via the World Wide Web.

3.2. Aims of Security

Many authors including Pfleeger (2000) and Pipkin (2000) describe three major of computer and information security, these being Confidentiality, Integrity and Availability. Two additional aims, Authenticity and Non-Repudiation are also often discussed. Essentially, any form of attack can be categorised as a breach of one or more of these aims.

This thesis will define these aims in the following manner:

> **Confidentiality:** This aim encompasses the idea that information or information systems should only be available to those that are authorised to access the resources.
>
> **Integrity:** Refers to the concept that data, information or information systems should be modified only by those that are authorised to do so.
>
> **Availability:** This aim suggests that information, systems or other resources should be available to authorised parties when required.

Authenticity: This aim states that people or devices must be correctly identified and determined to be genuine.

Non-Repudiation: The goal of non-repudiation is that entities must be accountable for the actions and be unable to falsely deny these actions.

This thesis takes the view that confidentiality, integrity and availability are the primary goals of any computer or information security effort and that authenticity and non-repudiation, while being important in their own right, support these first three aims. For example, the principle of confidentiality requires that only authorised people are able to read, view or make use of information. Authenticity plays a major role in the fulfilment of this aim, as it also does with integrity and availability. For this reason, this thesis refers to the three major aims of computer and information security and confidentiality, integrity and availability, while recognising the importance of authenticity and non-repudiation.

This thesis considers these aims as they relate to desktop systems. While these same aims can apply to a range of information assets and systems, this thesis is primarily concerned with desktop systems.

3.3. Threats Posed by Malicious Code

The idea of malicious code is nothing new. While some forms of malicious code did exist before the 1980s, that particular decade was pivotal in the history of malicious code. The late 1980s saw the emergence of several notable forms of malicious code including the Brain and Stoned viruses (White, Kephart, Chess, 1995) as well as the Morris Worm. These and other examples of malicious code demonstrated the vulnerability of systems to executable code written with malicious intent. While many of these examples affected the integrity and availability of systems and information, the potential was there for code to breach all of the aims previous mentioned.

The 1990s saw several new types of malicious code including macro viruses. These viruses forced many to re-think their views of viruses. These viruses propagated by attaching themselves to documents rather than executable files or boot sectors of disks. This proved to be quite a successful vector for virus propagation. Given the number of macro-supporting documents written, stored and shared, the use of documents as hosts for viruses led to many widespread infections (CERT, 2000a).

Since the late 1990s there have been a number of worms that have caused widespread infections. Some of these such as the Loveletter worm (CERT, 2000b) have blurred the lines between viruses and worms. Some have begun to refer to such pieces of malicious code as Virus/Worm Hybrids (Nachenberg, n.d).

Many of the forms of executable code in use today are quite different to those used in the 1980s and other periods in the history of computer usage. Today executable code may exist in the form of executable program files, document macros as well as other forms such as executable web content. One of the main aims of this thesis is to examine the possibility of malicious code being implemented using executable web content technologies such as Java and AvtiveX

3.4. Executable Web Content Specific Threats

This thesis identifies a number of risks and threats as being specific to executable web content. While the threats raised by forms of malicious code such as viruses, worms and Trojans have been clearly documented over a number of years, this thesis will expand upon some of these threats and will contend that there are several risks that specific to forms of executable web content such as Java and ActiveX.

3.4.1. Models of Code Distribution

Traditional file infecting and boot sector viruses such as Brain and Stoned achieved widespread infections due to the fact that their method of propagation mirrored the prevailing model of code distribution. At the time, sharing of executable code via the swapping of disks was common.

In more recent years email has been a major vector for infection by malicious code. Examples of email-borne viruses and worms such as Melissa and LoveLetter have highlighted the suitability of email as a major vector for malicious code attacks. Again these forms of malicious code have exploited a major mechanism for the distribution of executable code. The transferral of executable program code and macro capable documents is now so common that many forms of malicious code now use this as the primary method of propagation.

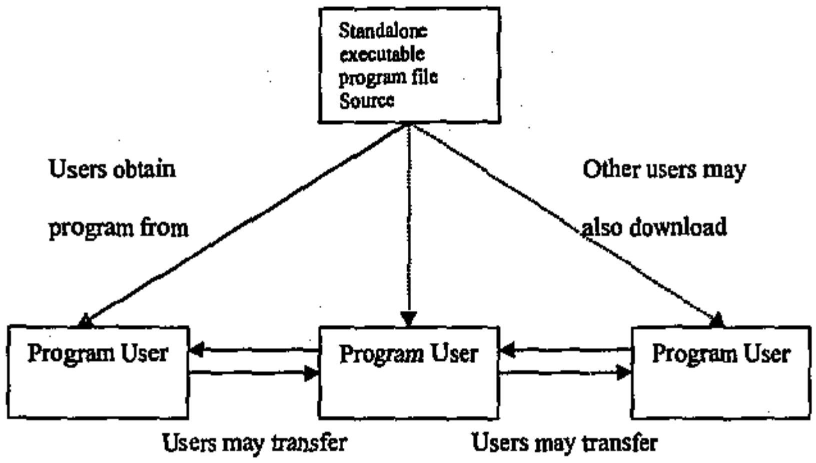
Executable web content employs a significantly different model of execution when compared with other forms of software. Such code is not distributed as a shrink-wrapped retail product, nor is it passed around between users, nor is it transferred via email. By definition executable web content is executable code that is attached to web pages and transparently downloaded and executed as part of that web page. As a result malicious executable web content will have significantly different vectors for infection than other forms of malicious code such as viruses and Trojan horses. As there is little sharing of Java Applets or ActiveX Controls directly between users (See Figures 1 & 2) the distribution models for viruses and executable web content are not very closely aligned.

The model of distribution of executable web content is more closely aligned with the typical distribution model of Trojan Horses. In this model of distribution, the malicious code is more likely to be distributed from a single source or group or sources than by propagation between users as is the case with a more conventional virus.

This thesis contends that the behaviour of malicious executable web content is more likely to be comparable with that of Trojan Horses than viruses or worm. While this thesis does not dismiss the possibility that a malicious ActiveX Control or Java Applet could be used as a delivery mechanism for a more conventional virus or worm, it does take the view that malicious Java or ActiveX Code will be more likely take the form of a Trojan.

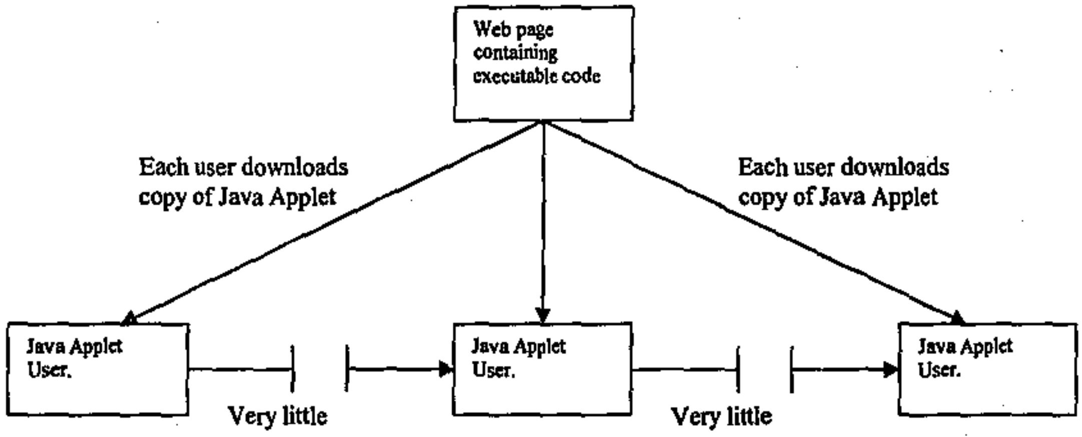
3.4.2. User Involvement

Another factor that distinguishes malicious executable web content from other forms of malicious code is the level of user involvement. As executable web content is run when the page containing it is viewed, there is often very little choice on the part of the user as to whether or not that code is to be executed. When the user makes a decision to go to a web page, there is no real prior indication that a page contains Java Applets or ActiveX Controls. In many cases if a user were to be affected by a piece of malicious executable web content, the only conscious decision might have been the initial decision to visit the web page. Depending of configuration of web browsers, personal firewalls, anti-malware or content filtering software, users may be presented with a warning prior to the execution of such code, at which point a conscious decision can be made. However in many cases the downloading and execution of the code happens automatically and transparently.



Standalone executable program file Source
Users obtain program from
Other users may also download
Program User
Program User
Program User
Users may transfer
Users may transfer

Figure 1: Standalone Program Distribution Model



Web page containing executable code
Each user downloads copy of Java Applet
Each user downloads copy of Java Applet
Java Applet User.
Java Applet User.
Java Applet User.
Very little
Very little

Figure 2: Executable Web Content Distribution Model

In this sense, executable web content is similar to executable code that might arrive via email. In both cases the user does not have to consciously seek out the piece of code.

3.5. Summary

Malicious executable code has the potential to affect the aims of confidentiality, integrity and availability. While it is not the intention of this thesis to re-examine the threat posed by malicious code in general, this thesis does contend that malicious executable web content does pose some specific threats. These threats have been described in this chapter. In particular this chapter has identified models of distribution and level of user interaction as two areas in which the possibility of malicious executable web content raises some specific concerns.

4. The Java Programming Language

4.1. Overview

When Sun Microsystems released the Java Language in 1995, it was surrounded by both genuine interest and a large degree of industry hype. Java is an object oriented programming language that is well suited to use with networked environments such as the World Wide Web.

Although Java can be used to develop stand-alone applications, much of its popularity stems from its networking capabilities. When used in an environment such as the World Wide Web, the Java language is typically used to create distributed applications referred to as applets. These applets can be downloaded and executed on a wide range of heterogeneous platforms. Java applets and ActiveX Controls (discussed later in this thesis) comprise two popular forms of executable web content.

This chapter will discuss the origins of the Java language, the characteristics that define it, its security architecture and the ways in which the language has evolved.

4.2. Introducing the Java Language

The Java programming language was developed by Sun Microsystems. The release of the language in 1995 was greeted with both gènuine interest and a high degree of industry hype. In many ways, the explosion of interest in this new language has mirrored the excitement surrounding the World Wide Web itself. In the years since its release, Java has become one of the most popular and high profile languages available to software developers (McGraw & Felten, 1998).

The Java language exists in several forms. While Sun distributes the language through various versions of its Java Development Kits (JDK 1.0, 1995; JDK 1.1, 1996; JDK 1.2, 1998), Java Runtime Environment (JRE, 1998) and other downloadable resources, Java technology has also been licensed by a number of vendors including Microsoft and Netscape. This thesis will use the term Java to describe the language as specified by Sun Microsystems and implemented in various versions of Sun's JDK.

There are a number of Java related technologies that exist around the periphery of the language itself. Some of these related technologies are produced by Sun Microsystems while others have been developed by other parties. It is not the intention of this thesis to examine all possible java-related technologies and APIs, rather it will discuss the basic language itself and the security issues that it raises.

One notable example of these peripheral technologies is what Sun has named Java Beans. Java Bean technology is an Application Programming Interface (API) that provides a software component architecture for the Java language (Hamilton, 1997). Java Beans are small, independent Java components that can be combined to create larger, more complex applications. Java beans have some similarities to ActiveX Controls in that they are both software component architectures.

There are also a number of other Java APIs that can be used with the Java language to provide database connectivity, speech capabilities, telephony features and other functions to extend the capabilities of the language (Sun Microsystems, 2000).

4.3. Java Vs JavaScript

It is important to note that Java and JavaScript are not the same things. JavaScript is a scripting language that can be used in conjunction with web pages to perform some actions when a page is viewed with a JavaScript capable browser. Unlike the Java language, JavaScripts are not compiled in any way. As stated earlier, this thesis intends to focus on binary forms of executable code and as such a detailed discussion regarding security issues raised by JavaScript and other scripting languages is outside the scope of this thesis.

Given the number of variations of the Java language, peripheral technologies such as Java Beans, and the number of additional APIs available, the terminology surrounding the Java language can become very confused. This thesis will use the term Java to describe the core language as specified by Sun Microsystems and as implemented in the various JDK releases.

4.4. Java Applets as a Form of Executable Web Content

Like most programming languages, Java can be used to create stand-alone applications. However, much of its popularity arises from its ability to create distributed applications referred to as applets.

These applets can be added to Web pages and as such, they comprise one form of executable web content. Java applets are typically downloaded to and executed on the client machine when the web page is viewed. When attached to web pages, Java applets can be used for a wide range of purposes. At one end of this spectrum, applets may be used to display simple eye-catching animations or perform other such tasks. Towards the centre of the spectrum, an applet could be used to extend the capabilities of a web page and/or browser, by adding user interface features. At the other extreme of this spectrum, Java applets could be used to deploy complex distributed applications.

A Java applet is added to a web page by using the APPLET tag within the HTML file that makes up the web page. As this thesis is not intended to act as an HTML reference, the syntax and semantics of the APPLET HTML tag will not be discussed here. However, detailed explanations and examples can be found in any number of HTML references or from organisations such as the World Wide Web Consortium (www.w3.org).

4.5. Characteristics of the Java Language

Sun Microsystems (1996) have described the Java language as "A simple, object-oriented, network-savvy, interpreted, robust, secure, architecture neutral, portable, high-performance, multithreaded, dynamic language."

In describing the language in such a way, it seems that Sun is acknowledging the fact that a large degree of hype surrounds the language. Regardless of this hype, this string of buzzwords does list some important characteristics of the language.

4.5.1. Portability

Portability is one of Java's most important characteristics (Sun, 1996; Gosling & McGilton, 1995). This portability has helped to make Java one of today's most popular languages. Given the portable, cross-platform nature of the language, Java is well suited to the heterogeneous nature of the World Wide Web and has become one of the most popular tools for developing distributed applications (McGraw & Felten, 1998).

Java's portability stems from its use of bytecodes as an intermediate level of compilation. Rather than being a completely compiled or completely interpreted language, Java takes a hybrid approach. Java source code is compiled to a series of byte codes, which are in turn interpreted by a Java Virtual Machine (JVM) (Pistoia, Relle, Gupta, Nagnu, Raman, 1999). The bytecodes comprise the instructions that drive the JVM. Theoretically, a JVM can be implemented as a piece of software running on almost any computer platform. Alternatively, a JVM could be implemented in hardware. In such a case, the Java bytecodes would form the native instructions for the Java machine. The difference would be largely transparent to Java program (Edwards, 1997).

While this "Write Once, Run Anywhere" (Sun, 1999) approach makes Java a viable alternative for many development projects, many consider that the performance degradation resulting from the interpretation process makes the language unsuitable for large complex applications. In many cases, Just-In-Time (JIT) compilers are considered necessary in order to improve the performance of Java code. Rather than interpreting Java bytecodes, JIT compilers compile the bytecodes into code native to the particular platform. This native code is generally faster to execute than interpreted bytecodes (Appel, 1999). In spite of the performance improvements offered by JIT compilers, there are still significant execution overheads compared to the execution of purely native code.

4.5.2. Security

Security is another important characteristic of the Java language. The security features of the language (which will be discussed later in this chapter) demonstrate some commitment on the part of Sun Microsystems to produce a secure language. Sun (1999) acknowledges that in a language as well suited to distributed computing, security is an important requirement. For this reason, security has been an important consideration since the earliest stages of the design of the language. In fact, it is often unusual for security to rank so highly as a consideration at such early stages of the development of the language (Pistoia et al. 1999).

The Java language boasts an integral security model (Pistoia et al., 1999; Gosling & McGilton, 1995), the evolution of which will be discussed in this chapter. This model has undergone several modifications since the release of the language in 1995. These modifications have been widely discussed by a number of commentators (Gong et al., 1997; Koved et al., 1998; Chess & Morar, 1998; McGraw & Felten, 1998). Each major revision of the Java language has seen significant changes to the security mechanisms offered by the language. The evolution of the Java security model shows an interesting progression away from an all-or-nothing approach towards a flexible, policy driven approach.

The cornerstone of Java security is a restrictive run-time environment commonly referred to as the Sandbox (Gong, 1998; Pistoia at al, 1999, p70). Since the release of the Java in 1995, the operation of this sandbox has evolved significantly with each revision of the language. Despite this evolution, its role has remained largely unchanged – to restrict the actions of untrusted, possibly malicious code.

4.6. The Evolution of the Java Security Model

The initial versions of the Java Language (JDK 1.0, 1995) provided a largely all-or-nothing approach to the issue of trust. The Java sandbox provided a tight restrictive environment in which untrusted applets could be safely executed. The decision as to whether or not an applet was considered to be trusted was made simply on the grounds of its source. Under this model, code loaded from the local file system would be considered to be trusted and would be allowed to operate without restriction. Alternatively, code loaded from external sources such as the World Wide Web would be subject to tight sandbox restrictions (Gong, 1998).

Comments were often made (Pistoia et al, 1999, p71), however that the tight sandbox restrictions of this initial model prevented reputable developers from fully exploiting the advantages offered by the Java language. In many cases, it was difficult to write practical software given the tight restrictions of the sandbox. Typically, untrusted code (any code not loaded from the local file-system) would not be allowed access to resources such as files. Additionally, applets would only be allowed to use network resources in order to contact the site from which the applet was downloaded. Chess & Morar (1998) also describes several other sandbox restrictions and in doing so, makes the point that it was inevitable that mechanisms would have to be provided to let trusted applets step outside of the restrictive sandbox.

Simply making more privileges available within the sandbox was not an adequate long-term solution (Presotto cited in Sun, 1996). There was a distinct danger that sandbox implementations would grow to include more and more privileges until the sandbox allowed almost full system access and restricted very little. Eventually this trend would defeat the purpose for which the sandbox was originally intended.

Sun's second major version of the Java Language (JDK 1.1, 1996) made some attempt to remedy this situation by allowing trusted applets to execute without the tight restrictions imposed by the sandbox (Pistoia et al., 1999, p72; Gong, 1998). While the sandbox remained an integral component of the JDK 1.1 security model, applets could now be signed using digital signature technologies. Applets with signatures trusted by the client were treated in much the same way as code loaded from the local file system, in that it would not be subjected to tight sandbox restrictions (Pistoia et al., 1999, p72). However, this was still largely an all or nothing approach. Decisions regarding trust were made on an applet-by-applet basis and an applet could only be considered either completely trusted or completely untrusted. Under this model, there was no notion that code could be partially trusted (McGraw & Felten, 1998).

The next version of the language saw several major changes. Not the least of which was a renaming of the language. With the release of JDK 1.2, Sun renamed the language Java 2. While the name Java 2 describes the current state of the language itself, the term JDK 1.2 is used to describe the Sun's implementation of this language. This thesis will adhere to this convention and use the term Java 2 to describe the language in general. The term JDK 1.2 will be used to describe a specific version of Sun's Java 2 implementation In addition to this change in name; JDK 1.2 introduced a heavily re-designed security architecture.

This latest security architecture focuses around the concept of a security policy, which can grant varying permissions to different applets in a fine-grained manner. In contrast to previous versions, this model does not force a yes or no decision to be made as to whether or not an applet is executed within the sandbox. Instead, applets can be assigned various privileges depending on the level of trust placed in the code. This highly flexible approach has the effect that, "the entire meaning of sandbox becomes a bit vague" (McGraw & Felten, 1998). Instead of one clearly defined sandbox, each applet can in effect, run in its own sandbox each of which can be afforded different permissions.

While this approach does offer a high degree of flexibility, it relies heavily on the creation of a sound policy. This raises the important of issue of who is responsible for the creation and maintenance of such a policy. End users may not have the experience or expertise necessary and system administrators may see such a security policy as a low priority in relation to other more pressing tasks (McGraw & Felten, 1998). In addition, once a policy is defined, it must be maintained. The environment in which such a policy operates is often very dynamic. There is a risk that once a policy is defined, it will be forgotten. In such a case, the old adage "out of sight, out of mind" may be particularly relevant.

This highlights the fact that the technical security features offered by the language are highly dependent on sound configuration. As such, management becomes a very important issue.

4.7. Key Components of the Java Security Model

There are several components of the Java language which enforce the Java Security model. In particular, Java makes use of what it refers to as the Bytecode verifier, Class Loader and Security Manager. Together these components work to enforce the Java Security Model.

Not surprisingly, the Java Class Loader is used to invoke Java classes as they are needed. A typical JVM contains a "Primordial" loader as well as any number of custom Class Loaders (Venners, 2002). These class loaders can if written appropriately, enforce separation of applets by providing distinct namespaces for different applets and applications (Venners, 2002; Oaks, 1998a). Class Loaders also aim to guard against malicious code masquerading as trusted Java APIs (Venners, 2002; McManis, 1996). Together with the Bytecode Verifier and Security Manager, the Class Loader comprise the major components of the Java Sandbox.

The Security Manager component enforces Sandbox restrictions by determining what actions can be taken by loaded classes. Oaks (1998b) makes the point that many of the restrictions enforced by the Security Manager are similar to the types of controls that one would normally consider to be the responsibility of an operating system, such as arbitrating access to files, network resources and other resources.

As the name suggests, the Bytecode Verifier is tasked with examining Java Classes to ensure that they conform to the specifications of the Java language. This aims to ensure that classes are not malformed either deliberately or accidentally. The Bytecode Verifier checks the integrity of the bytecodes to ensure that classes have not been created using hostile compilers, do not contain buffer overflows as well as performing many other tests. The eventual aim is that the once the class has been verified, it can be executed with confidence by the JVM (Gosling & McGilton, 1996).

4.8. Implementations of Java Technology

While this chapter has mainly discussed Sun's design of the Java language and its implementation via Sun's JDK releases, Java technology has been licensed by a number of vendors including Microsoft and Netscape. As a result, there are several major implementations of Java technology.

Many of today's major web browsers, including Internet Explorer and Netscape Navigator, Mozilla and Opera all support the use of the Java language. Java enabled web browsers often ship with their own implementation of the JVM. While each vendor supplied JVM should conform to the Java specifications from Sun, implementations can vary greatly. In addition, vendor supplied JVMs may incorporate proprietary extensions. As a result, it can be confusing as to which JVM is used to execute a particular piece of Java code. Additionally, it is reasonable to expect that different implementations may contain different bugs. These bugs could possibly be exploited in order to bypass various security mechanisms. In addition to the JVMs incorporated within web browsers, Sun distributes its own JVM as part of it JDKs and JREs. Sun also distributes Java 2 JVM plug-ins for the major web browsers. When these plug-ins are installed, older Java code can still be executed by the browser's inbuilt JVM. When Java 2 code is encountered, it can be diverted and executed using Sun's Java 2 plug-ins.

This situation can become very confused when multiple JVMs are installed on one machine. Chess & Morar (1998) describes a hypothetical case in which;

> "...you have a JVM developed by Microsoft inside Internet Explorer, a JVM developed by Netscape inside Navigator, a JVM developed by Sun inside Lotus Notes, and the Java plugins from Sun inside both browsers for a grand total of four different JVMs in five different locations using four different signature databases and four sets of security settings."

The replication of signature databases and security settings makes it very difficult to implement and maintain a coherent, overall security policy. Additionally there is the possibility that each different JVM will have its share of design and implementation errors, which may be exploited by an attacker. Currently there are no known tools to centralise the management of Java security across a number of separate JVMs (Chess & Morar, 1998).

4.9. Risks and Threats Associated with the Java Language

While Java is an interesting and no doubt useful technology, there are a number of threats associated with its use. As detailed in previous chapters, there are certain risks specific to executable code. As Java provides a means whereby executable code can be run on a client machine as a result of viewing a web page, its use does imply a certain level of risk.

4.10. Hostile Java Applets

Several hostile Java Applets have been written and for some the source code is available. Many of these act as Trojans with effects range from annoyances such as displaying images of Dancing Bears to the unauthorised use of resources such as power time to false login prompts designed to capture passwords (LaDue, n.d).

4.11. Challenges Facing the Java Language

While Java undoubtedly has a great deal of potential, there are a number of challenges facing the language. Concerns over the performance of the language, reluctance to rewrite legacy applications, problems with the "write once, run anywhere" concept and fighting between Sun and Microsoft all threaten the long term viability of the Java language.

The performance of the Java language has been seen by many as a major concern. Given Java's commitment to portability and its reliance on bytecodes as an intermediate level of compilation, it is inevitable that the performance of the language will suffer to some degree. One concern is that the Java language may not be able to offer the performance levels required for mission-critical applications. Sun claimed that the release of Java 2 would put an end to the performance problems that had previously plagued the language.

Another major concern is that the Java language is becoming fragmented as more and more platform specific APIs and class libraries become available. While such additions to the Java language can be helpful in optimising Java applications for particular platforms, they do tend to limit the "Write once, run anywhere" potential of the language. To combat this trend, Sun has implemented the "100% Pure Java" Program (Sun Microsystems, 1999). The aim of this program is to certify that a Java program does not rely on any platform specific code and that it has been tested for cross platform compatibility and portability.

Fighting between Sun and Microsoft also threatens the future of the Java language. Initially Microsoft licensed Java technology from Sun Microsystems. Sun considered that with Microsoft supporting Java, the language would quickly become a de facto standard. Microsoft was interested in licensing the language in order to compete with Netscape's Navigator browser, which also made use of Java (Wong, 1998).

Since its licensing of the Java language, Microsoft has been accused of trying to "kidnap" the Java language by distributing a JVM that Sun claimed violated Microsoft's license agreement. Sun claimed that Microsoft had deliberated attempted to undermine the cross-platform nature of the Java language; by adding platform specific APIs and omitting certain core Java APIs. As a result, Sun began legal action against Microsoft in October 1997 (Sun Microsystems, n.d; Microsoft Corporation, 1997).

While the Java language shows a lot of potential, its future is by no means guaranteed. While there is little doubt that the Java language has been thrust into public attention by some effective marketing, there is also a great deal of genuine interest surrounding the language. If nothing else, the language has highlighted the level of industry interest in a programming language suited to use with an environment such as the World Wide Web. The Java language has several important challenges ahead. How Sun handles these challenges will go a long way to deciding the future of the language.

4.12. Summary

The Java language was developed with several key objectives, including portability, robustness and security. Since the release of the language in 1995, Java has evolved significantly. The current version of the language is marketed under the name Java 2 and implemented by Sun in the form of the JDK 1.2.

Although its implementation and design may have changed, the Java's Sandbox remains a central component of the language. The purpose of this Sandbox is to restrict the actions of possibly malicious Java code, by executing this code within a protective run-time environment.

The most current version of the Java language allows various permissions to be granted to an applet depending on the level of trust placed in the code. These permissions can be granted in an applet-by-applet basis in accordance with a security policy. This provides a high level of flexibility but raises several issues regarding configuration and management.

5. Microsoft's ActiveX Architecture

5.1. Overview

The term ActiveX describes a number of technologies from Microsoft, all of which are based on the company's Component Object Model (COM) and Object Linking and Embedding (OLE) technologies. This thesis concentrates on one specific type of ActiveX Object - ActiveX Controls, as these can be added to web pages and comprise one popular form of executable web content.

ActiveX technology is tightly integrated with both the Windows family of operating systems and Microsoft Internet Explorer and as such, many of the security issues raised by the use of ActiveX will be discussed in later chapters. However, in order to fully understand these security implications, it is necessary to examine the architecture that underlies this technology. Some of the discussion in chapter does not relate directly to security issues, although a thorough understanding of the technology will enable a more detailed discussion of these issues in later chapters. This chapter discusses the architecture behind ActiveX and the security mechanisms put in place by the technology itself. Security mechanisms put in place by operating systems and applications that make use of ActiveX controls will not be discussed in this chapter.

5.2. ActiveX, COM and OLE

ActiveX is a term used to describe a range of technologies based on Microsoft's Component Object Model (COM) and Object Linking and Embedding (OLE) technologies.

Microsoft's Component Object Model (COM) is a specification designed to allow reusable binary objects to interoperate and communicate (Li & Economopoulos, 1997, p.11). As it is a binary specification, COM objects can be written in any programming language that can produce a binary result that conforms to these specifications.

Microsoft's Object Linking and Embedding (OLE) technology builds on the framework provided by COM. OLE's main role is to "enable and facilitate component integration" (MSDN [CD-Rom], 1997). OLE technology first appeared in 1991 and was originally designed as method for creating rich, compound documents that could incorporate a number of enhancements such as sound and video. The next version of OLE went way beyond this concept of compound documents and provided a much more comprehensive architecture for component integration (MSDN [CD-Rom], 1997).

Microsoft draws comparisons between the software component approach of COM/OLE and the hardware component approach of Integrated Circuits (ICs). Just as electronic devices can be created by connecting pre-made and pre-tested integrated circuits, component architectures such as OLE/COM allow software developers to create complex software by connecting existing components (MSDN [CD-Rom], 1997). Given that these components have been well tested and documented, software developers do not need to re-implement fundamental algorithms or even consider the implementation of the particular component.

In 1996, Microsoft coined the phrase ActiveX. This concept was intended to form the cornerstone of the corporation's "Activate the Internet" strategy. Microsoft drew together a range of concepts based on OLE and COM technologies and renamed them under the banner of ActiveX. While, the term ActiveX covers a range of objects including Automation Server and Controllers, COM objects, Documents and

Containers (Anderson, 1997, p. 9), this thesis concentrates on ActiveX Controls, as they comprise one form of executable web content.

5.3. OLE Controls and Visual Basic

Many Visual Basic developers would be familiar with the concept of component based software development, in particular with VBX and OLE Controls. Component based software development with Visual Basic began with the introduction of VBXs in Visual Basic 3. VBXs allowed software developers to create applications containing pre-built components and were essentially Windows Dynamic Link Library (DLL) files that conformed to certain architectural specifications. These components were usually self contained and controlled their own user interfaces.

While the original VBXs were a boon for software developers, they did have severe limitations. The specifications to which VBXs had to conform were limited to 16-bit Windows/Intel platforms. In order to be of use with operating systems such as Windows NT and Windows 95, a new 32-bit control architecture would need to be designed (Li & Economopoulos, 1997, p. 174).

This new 32-bit architecture took the form of OLE controls. These controls were considerably more powerful, flexible and robust than their VBX predecessors. In addition, OLE controls could be used by a range of containers other than visual basic (Li & Economopoulos, 1997, p. 174). OLE Controls are often referred to as OCXs as these controls were generally given this file extension.

5.4. Adapting OLE Controls to the World Wide Web

With a surge in the popularity of the World Wide Web, Microsoft attempted to prepare many of its existing technologies for use with this new medium. In the face of competition from technologies such as Sun's Java programming language, Microsoft made the decision to re-vamp its OLE control technologies in order to make them better suited to low-bandwidth Web usage (Li & Economopoulos, 1997, p 187).

Microsoft recognised that OLE Control-like components could be used to extend the capabilities of web pages in much the same way as with Visual Basic programs. However, with the low bandwidth environment of the World Wide Web the need for lean, efficient controls was even more pronounced than was ever the case with Visual Basic applications. This need gave rise to ActiveX Controls.

ActiveX Controls are effectively streamlined OLE Controls. While the specifications for OLE Controls require the control to implement a large amount of mandatory functionality, the requirements for ActiveX Controls are greatly relaxed. In order to qualify as an ActiveX Control, an object needs only to implement one mandatory interface (discussed later in this chapter). In addition, it must also be able to self-register and unregister (also described later in this chapter). This effectively means that any COM Object can qualify as an ActiveX Control, without having to fulfil higher-level OLE requirements. As a result, ActiveX Controls are free to implement only the interfaces are absolutely necessary. By freeing developers of the need to implement unnecessary features, ActiveX controls are better suited to use with the World Wide Web than previous OLE Controls (Microsoft Corporation, 1999).

ActiveX controls comprise one form of executable web content. Like Java applets, ActiveX controls can be added to web pages in such a way that they download, install and execute when the page is viewed with a compatible browser.

Microsoft's Internet Explorer browser has in built support for ActiveX controls. Plug-ins are available for Netscape that allow ActiveX controls to be used with Netscape Navigator.

As a specific type of ActiveX Object, controls are always in-process. That is they execute within the same process as their container application (Anderson, 1997, p10). When used as a form of executable web content, a Web Browser such as Microsoft Internet Explorer acts as the control's container. Hence, when a control is added to a web page it executes within the same process as the Web Browser.

5.5. Classifying ActiveX Controls

While they comprise one form of ActiveX object, ActiveX controls can be further divided into several categories. The main division revolves around whether a control is classed as visual or non-visual. Several control variations can be seen in Figure 3.

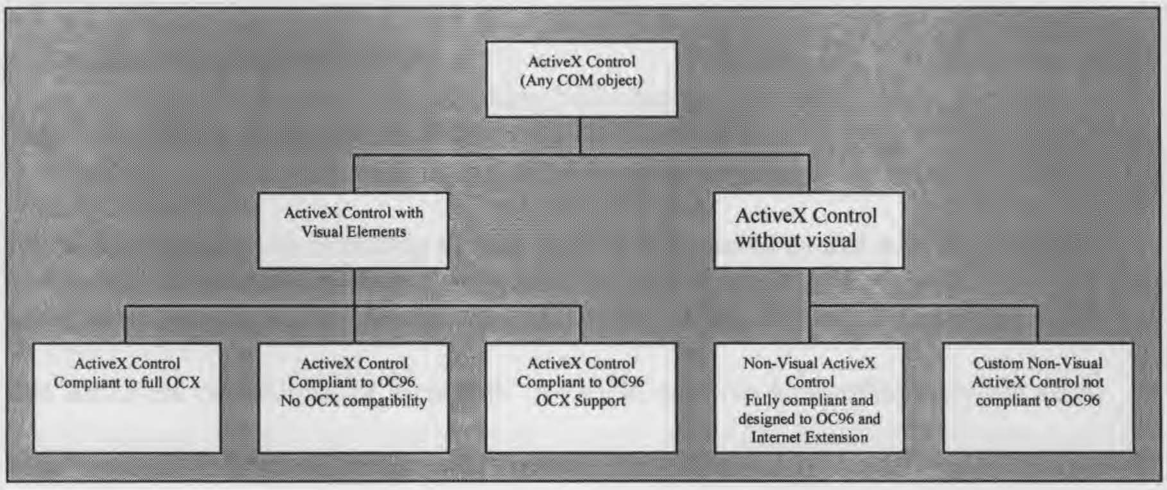

ActiveX Control (Any COM object)
ActiveX Control with Visual Elements
ActiveX Control without visual
ActiveX Control Compliant to full OCX
ActiveX Control Compliant to OC96. No OCX compatibility
ActiveX Control Compliant to OC96 OCX Support
Non-Visual ActiveX Control Fully compliant and designed to OC96 and Internet Extension
Custom Non-Visual ActiveX Control not compliant to OC96

Figure 3: ActiveX Control Types (Li & Economopoulos, 1997, p191)

ActiveX Controls can then be broken down into two categories – those with a visual representation and those without. Controls with visual representations are often used to extend the capabilities of a user interface. These controls not only manage their own data, they also maintain their own user interface (Li & Economopoulos, 1997, p192).

Alternatively, ActiveX Controls can exist without any form of user interface. These controls can be used to implement business logic or perform calculations behind the scenes. When embedded within a web page, ActiveX controls do not necessarily need to be a highly visible element of a web page.

5.6. ActiveX Control Capabilities

ActiveX Controls can make use of a variety of different Application Programming Interfaces (APIs), just as if the control were any other Windows executable program. As a result, ActiveX Controls can access a number of resources using standard Win32 functions including local file systems, network connections and the Windows registry.

Unlike with Java Applets, there is nothing built into the ActiveX Control architecture to restrain the actions of an ActiveX Control once it has begun execution. ActiveX Controls are subject to operating system security mechanisms and may be restrained using third party tools. However there is nothing in the ActiveX architecture itself that limits the capabilities of a control. As such, controls have effectively the same capabilities as standalone windows programs.

5.7. Implementing ActiveX Controls

ActiveX Controls are binary objects that conform to certain specifications. Historically, compiled OLE controls were given the extension .OCX, although they are effectively implemented within a Windows Dynamic Link Library (DLL). In fact, Microsoft now recommends that the extension DLL be used in favour of OCX (Microsoft Corporation, 1999). Not all Windows DLLs implement ActiveX Controls. Many simply implement libraries of compiled code. However, those that do contain ActiveX Controls may implement one or more controls within a single DLL file. More detailed discussion of the tools commonly used to develop ActiveX Controls can be found in Appendix B.

5.7.1. Interfaces and Methods

ActiveX Controls expose their functionality to the containers that host them through the methods that they implement. Related methods are usually grouped together to form interfaces. Each Interface of each COM object residing on a computer system has a unique Interface ID (IID).

Each ActiveX Control must implement at least one basic interface, commonly known as the IUnkown Interface. This interface contains three methods that are vitally important to the way in which COM Objects and ActiveX Controls operate.

The first of these three methods is called QueryInterface(). Programs making use of a COM object can use this method to obtain pointers to other interfaces implemented by the object. Client programs should only be able to access the functionality of an interface by first calling the QueryInterface() method (Li & Economopoulos, 1997, p. 28).

The other two methods of the IUnknown Interface deal with the fact that a COM object or ActiveX Control may be used concurrently by more than one client program. Each interface of a control contains a reference counter that determines when it is safe for a control to be discarded from memory. The AddRef() and Release() methods are used to increment and decrement these counters respectively.

When a client obtains a reference to an interface (by calling QueryInterface()) it must call the AddRef() method(Li & Economopoulos, 1997, p. 31). When the program no longer requires the services of the interface, it can call the Release() method to decrement the reference counter. When each interface is no longer needed, the control can unload itself from memory and free any resources that it currently holds (Li & Economopoulos, 1997, p. 32).

Many COM objects also support the concept of "late binding". Early binding is suitable if a client knows exactly what controls will be needed throughout the entire lifespan of the client application. In many cases, this is not practical, particularly in the case of development environments such as Visual Basic. The Visual Basic environment cannot reasonably be expected to know the details of every COM object it will ever host. Late binding solves this problem by allowing clients to discover the capabilities of a COM object at run-time rather than compile-time. Late binding is achieved through the use of a specific interface called IDispatch. This interface allows a client to determine the capabilities of an object at run-time (Li & Economopoulos, 1997, p. 54). So-called "Dual Interface" objects support both early and late binding (by providing an IDispatch interface), although there are significant performance overheads when late binding is used.

5.7.2. GUIDs and UUIDs

COM technology (and therefore ActiveX technology) relies heavily on the use of large, randomly generated numerical sequences. The generation process takes into account factors including the current date and time in order to produce a unique 128-bit identifier. The result is a randomly generated number large enough that the possibility of generating the same twice is negligible (Li & Economopoulos, 1997, p. 33).

These numbers are used to uniquely identify a range of entities including COM objects and the interfaces that they expose. The terminology used often differs depending on what it is that these sequences are identifying. When referring to COM/ActiveX technology several terms and abbreviations are commonly used. The main terms are summarised in Table 1.

Abbreviation	Term	Describes
GUID	Globally Unique Identifier	Used to describe a 128-bit identifier in general terms (not in any particular context).
UUID	Universally Unique Identifier	Used to describe a 128-bit identifier in general terms (not in any particular context).
CLSID	Class ID	Used to identify COM objects (Including ActiveX Controls.
IID	Interface ID	Uniquely identifies every interface implemented by every control.
CATID	Category ID	Identifies a component category. Used to state that a control implements certain functionality.

Table 1: Identifier Types

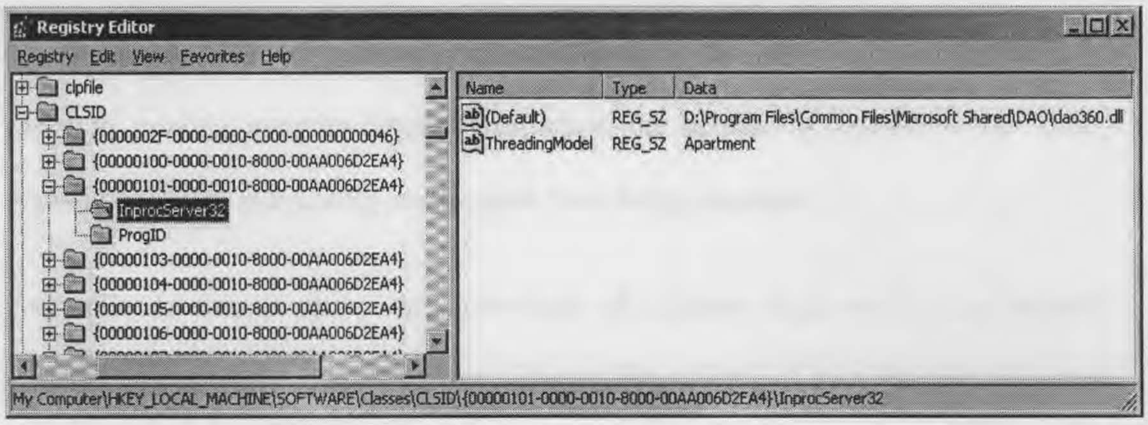
The use of such numerical identifiers eliminates the ambiguity that would be caused if such entities were simply assigned names. For example, by using unique identifiers, a program can be sure that it is using a particular COM Object rather than another entity that happens to have the same name.

5.8. ActiveX and the Windows Registry

ActiveX Controls rely heavily on the Windows Registry in order to operate (Li & Economopoulos, 1997, p. 46; Anderson, 1997, p. 35). The Windows Registry is a hierarchical repository containing a wide range of configuration data relating to the operating system itself as well as installed hardware and software and information regarding users. This registry is organised as an hierarchical collections of keys, sub-keys, values and data. The structure of this registry differs slightly depending on the version of Windows being used. However, at the top of the hierarchy are four main keys; **HKEY_CLASSES_ROOT, HKEY_CURRENT_USER, HKEY_LOCAL_MACHINE and HKEY_USERS.**

Each COM object (and therefore ActiveX Control) installed on a particular computer system has at least one entry in that system's registry. It is a requirement of an ActiveX Control that it be able to add and remove its own registry information (Anderson, 1997, p22; Li & Economopoulos, 1997, pp28-34). This is accomplished using two functions implemented within the control's .DLL file titled DLLRegisterServer and DLLUnregisterServer.

ActiveX Controls register themselves under the HKEY_CLASSES_ROOT key. This major key contains a sub-key labelled CLSID. The same registry information can be found under the HKEY_LOCAL_MACHINE/Software/Classes/CLSID. These two keys are functionally equivalent and can be used interchangeably. Each object registers its Class ID under this CLSID registry key. Each object can then add a number of sub-keys describing various properties belonging to the object. Several COM objects can be seen registered under the HKEY_CLASSES_ROOT/Software/Classes/CLSID key in Figure 4. This figure shows the 128-bit CLSIDs of several COM objects registered under the CLSID registry key. The figure also shows two sub-keys belonging to one COM Object and the values and data associated with the first of these sub-keys.



```
⊞ clpfile
⊟ CLSID
   ⊞ {0000002F-0000-0000-C000-000000000046}
   ⊞ {00000100-0000-0010-8000-00AA006D2EA4}
   ⊟ {00000101-0000-0010-8000-00AA006D2EA4}
        InprocServer32
        ProgID
   ⊞ {00000103-0000-0010-8000-00AA006D2EA4}
   ⊞ {00000104-0000-0010-8000-00AA006D2EA4}
   ⊞ {00000105-0000-0010-8000-00AA006D2EA4}
   ⊞ {00000106-0000-0010-8000-00AA006D2EA4}
```

Name	Type	Data
(Default)	REG_SZ	D:\Program Files\Common Files\Microsoft Shared\DAO\dao360.dll
ThreadingModel	REG_SZ	Apartment

My Computer\HKEY_LOCAL_MACHINE\SOFTWARE\Classes\CLSID\{00000101-0000-0010-8000-00AA006D2EA4}\InprocServer32

Figure 4: ActiveX Control Registry Information

The registry is vitally important to the operation of COM/ActiveX on Windows platforms.

5.9. ActiveX Controls and Microsoft Authenticode

ActiveX Control security is heavily reliant on Microsoft's Authenticode code signing technology. This section will discuss the underlying technology that powers Microsoft's Authenticode. However, it is often programs such as Internet Explorer that use this technology to provide security in relation to ActiveX Controls. Consequently, issues relating to the configuration, user interface and application of Authenticode technology will be discussed in conjunction with Microsoft's Internet Explorer Web Browser in later chapters. In contrast, this section will discuss the underlying concepts behind Authenticode and its relationship with ActiveX Controls.

ActiveX Control security revolves around users making an informed decision as to whether or not a control should be allowed to begin execution. Once a control has begun execution, the only restrictions placed on it are those provided by the operating system or other third party security tools.

This approach differs significantly with that taken by the Java language. While Java seeks to provide security through restricting the actions of applets at run time, ActiveX relies on preventing hostile code from being executed.

Authenticode aims to assure the authenticity of a binary object such as an ActiveX Control by positively identifying the author of the object. It also attempts to assure integrity by proving that control has not been modified since its release.

Authenticode 1.0 was released in 1996, and can be used to sign various forms of executable code including .EXE, .DLL, .OCX and Java Class files. While Authenticode can be used to sign a variety of types of code, it forms the only real line of defence against malicious ActiveX controls and therefore this technology is extremely important in relation to ActiveX.

5.9.1. Cryptographic Characteristics of Authenticode

Authenticode makes use of several existing technologies including X.509 certificates, PKCS #7 cryptographic standards and 1024-bit RSA keys for encryption and decryption (Feghhi, Feghhi & Williams, 1999, p. 102).

In order to sign code with Authenticode, developers must first generate a key pair and apply for a suitable certificate from a Certificate Authority (CA). In order to obtain a certificate, applicants must submit various personal details. The certificate, in effect binds this personal information to the developer's public key. Applicants must also agree to a pledge stating that they will not deliberately distribute code that is harmful or malicious in nature.

5.10. Security Concerns Surrounding ActiveX

The use of ActiveX Controls technology does give rise to certain security concerns. As this chapter has highlighted, ActiveX Controls are very powerful in that they execute directly on hardware and can make use of various libraries and APIs. Additionally they are not bound by any form of sandbox-like runtime restrictions.

This thesis contends that there are very real security concerns surrounding the use ActiveX controls as a form of executable web content. This section describes the security concerns identified by this thesis.

5.10.1. ActiveX Controls Can Be Very Powerful

ActiveX Controls can be very powerful in that they can make use of any number of libraries and APIs.

5.10.2. ActiveX Controls Do Not Execute within a Restrictive Environment

In contrast to Java Applets, ActiveX Controls are not designed to be executed within a restrictive run-time environment (CERT, 2000c). Once a Control has been allowed to begin execution, it is not restrained by any security measures other than those implemented by Operating Systems or third party products.

5.10.3. Reliance on Authentication

ActiveX Controls rely heavily on users being able to make decisions as to whether or not the control should be allowed to begin execution. Once a control has been allowed to begin execution, it can only be restrained through OS or third party controls. As a result, it is imperative that users can make a decision concerning the trustworthiness of the piece of code. This chapter has discussed the role played by Microsoft's Authenticode code signing technology in relation to ActiveX Controls.

This reliance on authentication does give rise to certain concerns. Firstly, there is a risk that users may not fully understand the consequences of allowing untrusted code to execute. The is often a tendency for users, when presented with a dialog box requesting permission for a Control to execute, to allow the action simply to dismiss the dialog box and continue. Often the consequences of allowing untrusted code to execute are not fully considered.

Additionally, reliance on authentication is a largely re-active stance. One aim of Microsoft's Authenticode technology is to allow legal action to be taken against individuals or organisations that sign malicious code. However, given the difficulties and expense involved with taking legal action against such individuals or organisations, often complicated by geographic, political and jurisdictional boundaries, such a re-active approach may not always be practical.

In an example highlighting the dangers of ActiveX's reliance on trust, The US Dept. of Energy sponsored Computer Incident Advisory Capability (CIAC) reports that in 2001, the certifying authority Verisign mistakenly issued two code signing certificates to an individual believing that the person in question was an employee of the Microsoft Corporation (CIAC, 2001). The certificates were issues the 29th and 30th January, 2001. The CIAC advisory issuing the warning was dated 22nd March, 2001.

Such certificates could have allowed the attacker to sign code including ActiveX Controls using the name "Microsoft Corporation". The code would not be automatically trusted, but by displaying the common name "Microsoft Corporation" the attacker could effectively be able to conduct a social engineering attack by convincing a user to allow the code to begin execution. As stated in the advisory published by CIAC (2001), "The danger…is that even a security-conscious user might agree to let the content execute, and might agree to always trust the bogus certificate".

When the mistake was discovered, Verisign revoked the certificates by adding them to the organisation Certificate Revocation Lists (CRLs). However the window between the issue of the certificates and their subsequent revocation could have given the attacker a substantial opportunity to use the certificates in a malicious manner. Additionally, as Versigin certificates did not specify a location for the CA's revocation list, web browsers were not able to verify the validity of the certificates once they had been revoked (CIAC, 2001).

Trust can be a complicated concept. While it is natural for users to associate the level of trust that they might have in a web page with the level of trust that they place in executable web content embedded in that page, such assumptions could be dangerous. For example, the author of a web page might not necessarily be the author of the controls used on that web page. It is not uncommon for web page authors to make use of third party controls. While the author of the web page might trust that the controls are free of malicious code, this might not be the case. Similarly, users of that web page may not draw a distinction between the page itself (which they may trust) and the executable code used on that page (which may come from a third party). Such users might not notice or be concerned about the fact that the common name in a certificate might not be the same as the common name of the site on which that code is hosted.

5.10.4. Controls Run with User's Permissions

All ActiveX Controls execute in-process; that is they execute within the same process as their parent container. When Controls are used in conjunction with web pages, this container is often a web browser such as Internet Explorer. As this container application executes within the security context of the current user, so too does the ActiveX Control (CERT, 2000c). As a result, if the current user has a high level of privileges so too will any ActiveX Control invoked by that user. If a user has access to various files, network resources so too will the ActiveX Control.

Additionally, if a malicious control is allowed to perform some kind of attack, any audit logs may identify the user that invoked the control as the source of the attack. As such, an unsuspecting user may be highlighted as the source of an attack.

5.10.5. Malicious Controls

There are definite concerns that ActiveX Controls could be used for malicious purposes. The most pressing concern in this area is that ActiveX Controls could be a very convenient mechanism for the delivery of a Trojan horse to a system or as a convenient delivery mechanism for a more conventional virus. As mentioned earlier, the distribution model used by ActiveX Controls

5.10.6. Exploitation of Legitimate Controls

In some cases it might not be necessary for an attacker to implement a malicious ActiveX Control. Attackers may be able to exploit vulnerabilities in existing controls using data driven attacks. Controls can be marked by their authors as being 'Safe for Scripting'. In effect, authors are claiming that their legitimate, non-malicious controls cannot be exploited by attackers using data driven attacks.

However this provides little assurance for control users. Organisations such as CERT have released a number of advisories that warn of controls that are incorrectly labelled as 'Safe for Scripting' (CERT, 1999b; CERT, 2000c; CERT, 2001).

5.10.7. Lack of Auditing and Management Tools

Windows does not have a log dedicated to downloaded code such as Java Applets and ActiveX Controls CERT (2000). Windows NT/2000/XP could be configured to audit modifications to certain registry keys (such as HKEY_CLASSES_ROOT_CLSID), however the volume of entries in this key could result in the generation of a large volumes log entries.

5.11. Summary

ActiveX is a term introduced by Microsoft to describe a variety of binary objects, all based in some way on COM/OLE technology. Of all the different types of ActiveX Components, this thesis is only concerned with ActiveX Controls, as they comprise one form of executable web content.

ActiveX Controls resulted from Microsoft's attempt to adapt OLE Controls to use with the World Wide Web. In a low-bandwidth environment such as the World Wide Web, it is necessary that controls are as lean and efficient as possible in order to reduce download times. For this reason, ActiveX Controls do not need to implement as much mandatory code as full OLE Controls. ActiveX Control developers need to implement very little mandatory code and are therefore free to implement as much or as little code as is necessary to solve the problem at hand.

ActiveX Controls can be classified as either visual or non-visual. Visual controls are often used to extend the user interface of their client Non-visual controls are well suited to implementing business rules and logic. ActiveX Controls can make use of a variety of APIs and have essentially the same capabilities as standalone executable programs.

Unlike Java Applets, ActiveX Controls are not restrained by any restrictive run-time environment. ActiveX Control security depends upon users making an informed decision as to whether or not a control should be allowed to begin execution. Microsoft's Authenticode code signing technology aims to prove the authenticity and integrity of ActiveX Controls as well as .DLLs, .EXEs, .OCXs and .CAB files and Java Applets.

ActiveX Controls and COM Components in are tightly integrated with both the Windows family of operating systems and Microsoft's Internet Explorer Web Browser. As a result, many of the security issues relating to ActiveX Controls will be discussed in relation to both Windows and Internet Explorer in later chapters.

6. The Role of Web Browser and Operating System Level Controls

6.1. Overview

This chapter examines the role of web browsers, operating systems and third party tools in controlling the actions of executable web content. While this thesis has argued the importance of security mechanisms that are integrated into executable web content technologies, this chapter highlights the importance of a layered of defence against the possibility of malicious web content.

6.2. Web Browser Issues

As executable web content technologies are closely integrated with web browsers it is important to consider the role of these applications in the execution and control of such code. While many issues make it difficult to get meaningful statistics on web browser usage, much of the web browser market is currently dominated by Microsoft's Internet Explorer product (BrowserNews, 2002; NUA Internet Surveys, 2002). However, there are a number of other browsers that also deserve some attention. Browsers such as Netscape, Mozilla and Opera still have a loyal following and it is important not to overlook these products.

6.2.1. Microsoft Internet Explorer

Microsoft's Internet Explorer product currently dominates the web browser market (BrowserNews, 2002). While it is acknowledged that Internet Explorer does suffer from a number of vulnerabilities and that such vulnerabilities continue to be found, this thesis limits its examination of this browser to the security mechanisms that it implements, particularly as they relate to issues involving executable web content.

6.2.2. Zones

Internet Explorer employs a concept of zones in order to classify web sites and pages and handle various forms of content accordingly. Internet, Local Intranet, Trusted and Restricted comprise the four zones provided by the browser and a set security controls can be applied to individually to each zone. Among these settings are options dealing with executable web content such as Java and ActiveX.

ActiveX related settings include options regarding the download and execution of signed and unsigned controls. One of the major concerns regarding executable web content raised in this thesis is that such the execution of such code is largely transparent to users. These web browser settings can alleviate this concern to some degree as the browser can be configured to prompt users for decisions regarding the downloading and execution of ActiveX Controls. Such prompting can however be seen by users as an annoyance, particularly as some websites might contain a number of ActiveX Controls, which would each prompt the user for a decision.

Zones can be configured to allow or disallow the downloading of both signed and unsigned controls, or to prompt the user for a decision. While the presence of such a signature can provide some degree of trust it does not completely guarantee that a control is non-malicious and safe for execution.

Another of the ActiveX related settings implemented by Internet Explorer determines the behaviour of the browser when confronted with controls that are marks as being "safe for scripting". This is intended to protect against situation in which an attacker might use scripts to control existing ActiveX controls and use them in a hostile manner. In this type of attack, the control itself is not malicious, although the attacker tries to use the control in a manner that is. By being marked as safe for scripting, the control is effectively claiming that it cannot be exploited in this fashion. The "Safe for Scripting" security setting offered by Internet Explorer governs whether such controls are initialised automatically, prevented from initialising or whether a user prompt is issued.

There are also a number of settings that govern the way in which Internet Explorer interacts with its Java Virtual Machine (JVM). A number of pre-defined security levels can be invoked for Java Applets, or settings can be customised. As the JVM has the ability to restrict code once it has begun execution, the list of custom settings offered by Internet Explorer are quite extensive when compared to the settings controlling ActiveX Controls. The browser can work in conjunction with the underlying JVM in order to control capabilities of Java Applets, whereas with ActiveX controls the decisions revolve around deciding whether or not a control should allowed to begin execution.

While the ability to classify web pages and sites and configure a range of security settings is a positive attribute of the Internet Explorer browser, the effective of such an approach does rely heavily on its configuration. An administration kit from Microsoft is available which allows administrators to configure these settings across a range of individual installations in a consistent manner. More detail regarding Internet Explorers Zone Settings can be found in Appendix C.

6.2.3. Third Party, Internet Explorer Based Browsers

There are a number of web browsers based on Microsoft's web browser engine that forms the basis of Internet Explorer. The majority of such browsers offer identical security features to Internet Explorer and will not be discussed in detail in this thesis. Some of these browsers do differ slightly in terms of cookie handling and other such functionality and there is the possibility that such browsers may have design flaws, implementation flaws and other vulnerabilities not found in Internet Explorer.

However it is not the intention of this thesis to examine Internet Explorer based browsers in detail.

6.2.4. Netscape/Mozilla

Despite the early popularity of the Netscape web browser, Microsoft's Internet Explorer currently enjoys dominance on Windows platforms (BrowserNews, 2002; NUA Internet Surveys, 2002). However browsers other than Internet Explorer are used on Windows platforms.

It is important to note that current versions of the Netscape browser are actually based on the Mozilla Web browser. As a result Netscape versions 6.0 and higher are quite different to earlier version of Netscape. This thesis will discuss Netscape and Mozilla as being essentially one product.

Netscape and Mozilla are available for several platforms including Windows, Macintosh and Linux. Mozilla does support Java, although it does not natively support ActiveX Controls. However plug-ins did exist that allowed earlier versions of Netscape to use ActiveX Controls. These plug-ins also work with Mozilla and another project to add ActiveX Support to the browser (albeit in a rather limited fashion) is currently underway (Lock, 2002).

As a result of Mozilla's lack of integrated ActiveX Control support, the browser's executable web content security is largely limited to sandbox restrictions enforce by the Java Virtual Machine. The browser does have an option to enable or disable JavaScript and there are some cookie management features. While the concept of Internet Explorer-like zones is not as important in a browser like Mozilla that does not support ActiveX, the lack of such features does limit the user's control of

JavaScripts. The ability to enable or disable JavaScript on a site-by-site basis would be a welcome addition.

6.2.5. Opera

Opera is another alternative to the Internet Explorer browser. Like Mozilla, Opera is available for a number of platforms including Windows, Macintosh and Linux. As a result of this cross platform nature, Opera also does not support ActiveX Controls. Hence executable web content security is largely enforced by the Java sandbox. The browser provides simple options for enabling and disabling Java and JavaScript. As with Mozilla, this approach lacks the fine grained control of web elements such as JavaScripts that can be achieved through the use of Internet Explorer's Zones.

6.3. Operating System Issues

This section will endeavour to highlight the importance of operating system level controls when dealing with executable web content. This thesis presents the view that operating system level controls are an important part of a layered defence when dealing with possibly malicious executable web content technologies, although they do not provide a complete solution to the problems raised.

This thesis will demonstrate that operating system level controls alone do not address the problems associated with malicious code, as there is not one standard set of security functionality that is provided by all operating systems that might encounter such code. The controls offered by Windows NT/2000/XP are very different to those offered by Windows 95/98/98SE/ME. These are quite different again when compared to Unix and Linux machines and Macintosh systems. As these operating systems do not implement a standard set of security features there is most definitely a role to be played by executable web content technologies themselves.

This chapter will focus on Windows NT/2000/XP and examine the security controls that are provided by this family of operating systems. A more detailed discussion of this family of operating systems can be found in Appendix D.

6.3.1. File Permissions

Unlike operating systems such as Windows 95/98/ME, Windows NT/2000/XP provides quite robust file permission functionality. Such functionality is available when the NTFS file system is used.

File permissions prevent users from interfering with files owned by other users. Under the Windows NT/2000/XP architecture, executable web content executes with the security context of the current user. Therefore file permissions could be used to prevent malicious code executed by one user form interfering with the files belonging to another user. However such controls would not prevent the malicious code from interfering with files owned by the current user. As such, some benefit may be gained may be gained from the use of file permissions when multiple users have files on the client machine or network shares accessible on the client machine. This assumes that the NTFS file system is used and that file permissions have been set. There is little protection to be gained from operating system level controls on files belonging to the current user as any malicious code would be executed with the permissions and privileges associated with that user.

6.3.2. Cryptographic Separation

Windows 2000 and Windows XP offer an encrypted file system that can be used to encrypt files. This prevents information being disclosed in the event of the theft of a hard disk, or by the attacker booting another operating system and using tools such as NTFSDos to gain read access to NTFS volumes. However such measures would be ineffective against malicious code such as ActiveX Controls as such code would execute with the permissions of the current user. This user would be able to decrypt file from the file system. In this sense, a transparent, encrypted file system such as the one offered by Windows 2000 and XP will provide no more protection than that provided by file permission mechanisms. Malicious ActiveX Controls would not be able to decrypt files belonging to other users, however file permissions could also be used to restrict such access.

6.3.3. Logging and auditing

Windows NT, 2000 and XP provide the ability to log a range of different events, including file accesses and uses of certain privileges. Three main logs are managed by these operating systems; a system log, an application log, and a security log.

6.3.4. Logging ActiveX Controls

Operating Systems such as Windows NT/2000/XP provide important logging and auditing features. These features can be used to record file accesses, successful and unsuccessful uses of privileges as well as errors and warnings. This auditing is performed by Security Reference Monitor and the Local Security Authority components (See Section 13.6, in appendix D). Windows 95/98/ME does not include such functionality.

The addition and removal of ActiveX Controls could be logged by auditing accesses to the HKEY_CLASSES_ROOT key or the HKEY_LOCAL_MACHINE/Software/Classes/CLSID key. However, given the large number of keys placed under these keys, such auditing may impose a significant performance overhead.

Some utilities are available that allow users to view registered ActiveX Controls including the OLEView tool from Microsoft (http://www.microsoft.com/Com/resources/oleview.asp#OLEViewer). However, due to the extensive use of OLE and COM technology within the Windows family of operating systems, many such tools display large numbers of objects, many of which are operating system components rather than installed web content.

It would be advantageous to be able to record the addition and removal of executable web content, in particular ActiveX Controls, in an Internet specific log. While logging and auditing are largely re-active measures, such a log would be a welcome addition.

6.4. Third Party Tools

While it is important to consider controls implemented by executable web content technologies themselves as well as operating system level controls, third party tools can also play an important role in protecting against malicious executable web content. This section will examine the role of tools such as personal firewalls and web content filters.

6.4.1. Anti-Malware Tools

Of all the types of third party security tools mentioned in this section, Anti-virus tools are probably the most well known. However this thesis will use the term Anti-Malware software to describe such products in order to reflect the fact that modern anti-virus software protects against more than just viruses. Such products typically provide protection against, viruses, Trojans, worms and in some cases malicious web content in the form of Java applets and ActiveX Controls.

6.4.2. Personal Firewalls

Personal Firewalls are similar to network firewalls in that they can apply filtering to network communications. However, personal firewalls are software products that operate on client machines. Some personal firewalls, including Norton Personal Firewall and Outpost can be used to filter out Java applets and ActiveX Controls, or at least prompt users for decisions as to whether or not these types of code should be allowed to begin execution.

It is important to note that ActiveX Controls in particular execute within the same process as the web browser that is hosting it. While many personal firewalls apply controls on an application-by-application basis, a malicious ActiveX Control acting within the process of a web browser, would appear to that personal firewall to be the web browser itself. As such, users may elect to trust the web browser, yet malicious code in the form of an ActiveX Control could exploit this trust and perform malicious actions.

6.4.3. Web Content Filters

Web content filtering tools can be used to guard against malicious executable web content as well as for a range of other purposes. Among other things, web content filters can be used to filter out Java applets and Active Controls. Some tools implement lists of trusted an un-trusted sites and allow a policy to be implemented accordingly. Such a policy might include the filtering of Java applets and ActiveX Controls.

Some tools such as Naviscope (2001) and Web-washer (2002) operate as personal proxy servers. Like their more fully fledged relatives, personal proxies operate as intermediaries between web clients and web servers. However, personal proxy servers reside on the same machine as the web client.

The use of such trusted and un-trusted lists is similar to the concept of zones implemented in Internet Explorer. Effective use of such zones in Internet Explorer would render such third party proxies redundant. However, such tools can be useful when browsers without the functionality of Internet Explorer's zones are used. Such proxies can also be useful when multiple browsers are installed on the one machine. A personal proxy server could be used to apply a consistent executable web content policy despite a particular user's choice of browser.

6.4.4. Cryptographic Tools

Third party cryptographic tools could provide some protection for sensitive files from malicious web content. Unlike a transparent, encrypted file system, the use of third party software to manually encrypt and decrypt sensitive files could prevent theft of information by code such as malicious ActiveX Controls.

6.5. Summary

This chapter examines the role of web browser, operating system and third party tools in protecting against malicious web content such as Java applets and ActiveX Controls. While these layers of protection are significant and play an important role, they do not diminish the need for controls to be implemented by the technologies themselves. Technologies such as Java may be used across a range of platforms, operating systems and web browsers. While ActiveX Controls are more Windows oriented, they can also be used across a range of web browsers and Windows

Platforms. In both cases there may be a very wide range of third party security tools in use.

This chapter highlights one of the main distinctions between Java and ActiveX. ActiveX's reliance on code signing and lack of sandbox-like run-time environment increases the reliance of users on browser, OS and third party level controls. However as such code executes within the security context of the current user, files and other resources belong to that user may by at risk.

7. Comparison and Evaluation of Security Architectures

7.1. Overview

Technologies such as Java and ActiveX fill a similar niche. They both provide a mechanism whereby web developers can extend the capabilities of web pages and work around limitations of HTML. While this is not the only application of these technologies, it is one area where there is a definite overlap between the two.

It is true that both Java and ActiveX have a very different design philosophy, security architecture and method of implementation, however comparisons between the two are inevitable. The terms Java and ActiveX are often used in the same context. Where people refer to one, they often make mention of the other. When Antivirus software provides functionality to verify one it usually does so for the other as well. When a personal firewall allows the blocking of one it usually does so for the other as well.

This chapter compares, contrasts and comments on the security architectures, models and implementations of these two technologies. In particular, it pays attention to the very different approaches to issues of security offered by the two technologies. It contrasts the sandbox approach of java, with ActiveX's reliance of code signing.

7.2. Evolution Vs Revolution

Previous chapters have made mention of the design philosophies behind Java and ActiveX and the origins of both of these technologies. ActiveX is the result of an evolutionary process that began with VBX controls and OLE objects. While existing languages influenced the design of the Java language, it was the result of a specific design process rather than an evolution from previous products.

This allowed the designers of the Java language to consider code security as one of the major design goals of the language. While other issues such as portability and robustness were also important design considerations, it was quite unusual for a security model such as the one implemented by Java, to be considered at such an early stage of development and so tightly integrated into the language. In contrast, ActiveX evolved from an environment in which code integrity and security was not such an important issue.

7.3. Security Models Vs Trust Models

As mentioned in previous chapters, Java employs a highly integrated security model that encompasses both authentication and authorisation. In the later versions of the Java language, authentication can be achieved through the use of digital signatures and authorisation can be enforced by the Java sandbox.

In contrast, ActiveX relies on verification of integrity and authenticity through code signing. ActiveX lacks any method to enforce controls over what a control can do once it has been allowed to begin execution. Operating system controls can offer some protection particularly when multiple users share the same machine and controls are enforced via file permissions. As the ActiveX control operates within the security context of the user that launched the browser. Additionally, third party products such as encryption tools may offer some protection against theft of information attacks that could be performed using ActiveX controls. However the fact that ActiveX technology does not provided any integrated mechanism to control the activity of controls is a major concern.

It could be argued that stand-alone executables do not provide an in-built security model and that therefore this omission is from ActiveX technology is not an important issue. However this thesis argues that executable web content is designed to integrate seamlessly with web pages and is often quite transparent to users. This removes the need for a user to consciously and explicitly, seek out, download and execute code (which could possibly be malicious in nature). As a result, this thesis argues that there is some responsibility for executable web content technologies to implement controls that can restrict the actions of a piece of code. Java makes a well-intentioned, reasonable attempt to provide such a mechanism through is sandbox approach. ActiveX makes no such attempt.

While Java's sandbox approach does attempt to provide a safe, restricted run-time environment for executable web content, its developers have, in the past, struggled to define the boundaries of this environment. As mentioned in previous chapters, Java's security model has undergone significant changes. The initial release of the Java language saw a largely all-or-nothing security model under which all local Java applications were completely trusted an allowed to operate without restriction whereas remote applets were subject to significant sandbox restrictions. Since this initial release there has been a distinct move away from this all-or-nothing approach, to a more fine-grained, policy driven arrangement. The latest versions of the language allow sandbox restrictions to be tailored for specific applets based on a security policy.

While the developers of the Java language should be commended for firstly designing the language with a tightly integrated security model and then for refining this model, there are concerns that the policy driven approach may be self-defeating in its complexity. Referring to the policy driven approach of Java 2, Schneier (2000, p167) states "This works much better, but has proven too complicated to use".

Commentators such as Bruce Schneier (2002b) have raised a number of concerns regarding code signing as a means of protection against malicious code. Schneier (2002b) cautions, "Remember, digital signatures provide accountability, not protection." and also makes the point that "Code signing can't protect you if you can't figure out whom to trust".

7.4. Implementation Issues, Bugs and Vulnerabilities

This is one area of concern, particularly with the Java language. As noted earlier, there are a number of Java Virtual Machine implementations from many vendors. While all of these implementations should conform to the Java specifications, it is reasonable to expect that there will be a number of vulnerabilities that could potentially be exploited.

Not surprisingly, since the release of the Java language in 1995, a number of significant vulnerabilities have been found. Sun Microsystems maintains a chronological list of such bugs (Sun Microsystems, 2002). Examining this list tends to emphasise the fact that different implementations will have different vulnerabilities and flaws.

The most recent example documented on this list describes a possible attack to escalate the privileges of a piece of Java code by exploiting a vulnerability in the Bytecode Verifier of the Java Sandbox. However privilege escalation attacks are not the only type of problem documents. Attacks against confidentiality and availability of information and systems can also be found in this list.

7.5. Executable Web Content Security

In a paper titled *A Comparison between Java and ActiveX Security*, Hopwood (1997) asks the question "Would ActiveX or Java be secure if all implementation bugs were fixed?" While the security architecture of Java in particular has changed significantly since Hopwood wrote this paper, it remains an interesting question as it highlights the differences between the design philosophy and security architectures of the two technologies.

7.6. Summary

This chapter compares and contrasts the approaches taken by the developers of the Java and ActiveX technologies. It builds on previous chapters and argues the merits and weaknesses of the approaches taken by these technologies to the difficult task of executable web content security.

This thesis argues that there are inherent risks involved with the principle of attaching executable code to web pages in such a way that they download and execute transparently on client systems. It is therefore important to address these issues and consider the security models implemented by technologies such as Java and ActiveX.

This chapter argues that the approach taken by the Java language at least attempts to address the inherent risks associated with executable web content, while ActiveX's reliance on digital signatures does little to address these concerns.

8. Research Questions

8.1. Overview

This chapter provides answers to the research questions identified in Chapter 2. It provides and analysis of chapters presented in this thesis and aims to provide clear, concise answers to these questions.

8.2. Does executable WWW content pose a significant security threat to client machines?

This thesis argues that there are inherent risks associated with the use of executable web content technologies such as Java and ActiveX. Security problems associated with untrusted, potentially malicious code have been well documented over a number of years. However with most forms of executable code, there is a conscious decision on the part of users to first seek out, download and then execute the code. This is not the case with code embedded in web pages.

Web users will not necessarily be aware that a web page contains executable code before visiting that page. This, combined with the fact that such code could be downloaded and executed in a largely transparent manner, removes much of the decision making from the user.

8.3. Do the security mechanisms offered by these technologies provide a suitable level of protection?

Both of the major forms of executable web content discussed in this thesis implement some form of security or trust mechanisms. However there are stark differences between the approaches taken by Java and ActiveX.

Java's sandbox approach acknowledges some the concerns raised by the use of executable web content by providing a mechanism with which to restrict the capability of a piece of code. This sandbox approach has a number of positive attributes.

Java has been designed as an architecture neutral language. It is intended that applets can be written once and then executed on a number of very different platforms. The Java Virtual Machine is the cornerstone of Java portability. As such, it would be inappropriate for the Java language to rely on operating system or other platform specific controls. The large variation in security controls offered by various operating systems necessitates a security model that is integrated into the language itself.

In contrast ActiveX is limited to Windows platforms. Despite this, ActiveX technology cannot rely on a certain set of operating system security features being present. The Windows 9x product line and the Windows NT/2000/XP line provide a very different set of security functionality. As such the ActiveX technology can not rely on the presence of certain OS level controls. For this reason, this thesis argues that ActiveX provides insufficient protect against the threats raise by the use of such code.

8.4. Are there significant differences in the security mechanisms provided by popular WWW browsers?

The significance of this question has changed somewhat during the writing of this thesis. The current dominance of Microsoft's Internet Explorer web browser has reduced the importance of this question as it is written. However the issue of web browser security mechanisms is still an important one.

Internet Explorer's concept of security zones is an important step. This feature does allow the implementation of a security policy in that web sites can be classified and that application level controls can be applied depending upon this classification. When Internet Explorer initially introduced the Security Zone functionality, its main competitors did not have any equivalent features.

There are some significant differences in the executable web content security functionality provided by the current popular browsers. This is to be expected as there are some quite fundamental differences in terms of the types of executable web content supported by such browsers. Internet Explorer's support for ActiveX does necessitate the concept of zones that is supported by the browser. Browser's such as Mozilla and Opera that do not support ActiveX (natively) can afford to rely on the security features offered by the Java sandbox approach for executable web content security. However, Internet Explorers Zones concept spread beyond ActiveX and allows users increased control over scripts and cookies.

The use of third party tools can play an important role in enforcing a consistent executable web content policy across a number of web browsers. Tools such as privacy enhancing proxy servers can be useful when multiple web browsers are present on one machine. By using tools such as these personal proxy servers, users can enforce a consistent policy regarding executable web content such as Java and ActiveX and scripts, as well as cookies and banner advertisements, regardless of the security features provided by web browsers.

8.5. Are there significant benefits to be gained from using secure desktop operating systems in conjunction with WWW applications?

This thesis argues that operating system level controls are an important factor when considering executable web content technologies. However as stated earlier, it is the position of this thesis that operating system level controls on their own are not sufficient, rather that they play an important role in terms of defence in depth. As argued earlier, technologies such as Java and ActiveX cannot assume that a certain set of operating system level controls will be present. ActiveX controls, while largely limited to the Windows platform could be expected to execute on Windows 9x systems or Windows NT/2000/XP systems. The situation is more complex in terms of Java applets, which could be expected to operate on Windows and Macintosh systems, as well as Linux and Unix variants.

ActiveX's reliance on digital signatures and assurances of authenticity and integrity result in a strong need for operating system level controls. As ActiveX controls operate in-process with respect to the web browser used, they operate with the same permissions as the user of the operating system. As such, when multiple users share systems, operating system level permissions are necessary to separate resources belonging to these users. While a malicious ActiveX control may be able to affect the resources to which the particular user has access, it should not be able to affect objects belonging to other users.

Systems such as Windows 9x machines are more problematic as far as ActiveX controls are concerned. A lack of strong operating system level resource permissions means that if allowed to begin execution, a control will effectively have unrestricted access to all of the resources available on that machine. In contrast, Java applets would still be confined by the restrictive run-time environment of the Java Sandbox.

It is the position of this thesis that operating system level controls are of great importance when considering the possibility of malicious executable web content, but as part of a defence in depth approach. This thesis contends that controls are necessary at the level of the technology itself, as well as the operating system level.

The controls implemented by the technologies themselves are often closely integrated with the application level controls such as Internet Explorer's Security Zone concept.

8.6. Summary

This chapter examines re-visits the research questions identified in Chapter 2. Perhaps this chapter should close with a statement made by Bruce Schneier (2002a) "Mobile code is very dangerous, but it's here to stay. For mobile code to survive, it should be redesigned with security as a primary feature."

9. Conclusions and Future Research

9.1. Overview

This chapter provides some concluding remarks and suggests possible areas for future research. A number of such areas were identified during the preparation of this thesis. Some of these areas are quite closely related to issues discussed in this document, but were considered to fall outside the scope of this thesis. Other topics such as peer-to-peer networking have been suggested as areas of future research due to their sudden prominence and widespread use. Given the sudden surge in use of peer-to-peer technologies, it will become increasingly important to be aware of the security issues surrounding their use.

9.2. Conclusions

This thesis has examined issues surrounding the use of executable web content and has examined the possibilities for malicious code to be delivered in this manner. In particular it has focused on Sun Microsystems' Java Programming Language and Microsoft's ActiveX Control Technology.

The general conclusions reached by this investigation are that there are significant risks inherent with the concept of attaching binary, executable code to web page in such a way that the code is automatically downloaded and executed when the web page is rendered within a browser.

The dangers of running code from untrusted sources have been well documented over a number of years. Throughout the last two decades in particular, the vectors for attack by forms of malicious code have mirrored the prevailing methods of code distribution. In the 1980s and early 1990s, file infecting and boot sector viruses were common. This mirrored the fact that code executable code was commonly distributed between users via floppy disks. During the mid to late 1990s, email became a major vector for malicious code attacks. Often this involved documents infected with macro viruses. Again, this mirrored the fact that email had become one of the major ways in which executable code was distributed.

Technologies such as Java and ActiveX represent another method for distributing executable code. This thesis takes the view that the distribution of such code via web pages represents another mode of executable code distribution and has the potential to become a major vector for malicious code attacks.

Technologies such as Java and ActiveX increase the possibility that users will execute code from untrusted sources. However, it is not entirely practical to advocate that such technologies are not used. Users tend to expect a certain amount of functionality from web pages and many services rely on embedded code. Internet banking and similar services often make use of these sorts of technologies. Simply advising web users to turn off Java and ActiveX is becoming less and less practical as more service begin to rely on such technologies. As a result it is important to understand the features and limitations of the security measures offered by such technologies.

This thesis takes the view that the security model offered by the Java Programming Language is a positive aspect of the language. The Java security model does not make any assumptions about the security capabilities of the underlying system and this tends to reflect the portable nature of the language.

In contrast, this thesis also takes the view that ActiveX's reliance on authenticity, integrity and non-repudiation through digital signatures raises some concerns. Once an ActiveX Control is allowed to begin execution it is only really constrained by operating system and third party controls. Given that the capabilities of such operating system and third party controls can vary from system to system, the effectiveness of this approach can vary dramatically.

This thesis contends that when considering the issue of executable web content, a layered defence must be employed. The first layer in series of defences should be available at the level of the technology itself. Java's sandbox model is an important step in this direction. Conversely, this thesis has some concerns over ActiveX's reliance on digital signatures.

Application level defences comprise the next layer in this series of defences. This thesis has examined the security mechanisms of several popular web browsers, including Internet Explorer, Netscape, Mozilla and Opera. It must be noted that this thesis has limited its examination of these browsers to the principles behind the security mechanisms implemented by these products, particularly as they relate to executable web content. It is acknowledged that many vulnerabilities have been and will continue to be discovered in various browsers. While many of these vulnerabilities could result in significant security breaches, a discussion of individual vulnerabilities is well beyond the scope of this thesis.

This thesis takes the view that flexibly policy based approaches such as that offered by the concept of Zones in the Internet Explorer range of web browsers is a positive step, even though this approach cannot restrain the actions of an ActiveX Control once it has begun execution.

Aside from web browsers themselves, application level controls might also include a number of third products such file encryption tools, personal firewalls, anti-malware as well as auditing and logging tools.

Finally, operating system level controls are also of great importance. This is one area that varies considerably between systems. For example, systems employing the Windows 98 operating system, will provide very different functionality to those employing the Linux, or Windows NT/2000/XP. This thesis contends that while operating system level controls are an important aspect of executable web content security, the variation in functionality offered by client operating systems indicates that other levels of controls will also be of great importance.

Finally, while they are outside the scope of this thesis, this author acknowledges the importance of non-technical measures such as education and awareness of end users as well as a solid policy framework, in which these users make use of World Wide Web resources.

9.3. Future Research

During the preparation of this thesis, it became clear that there are a number of World Wide Web and other Internet related security issues that that need attention. It was unfortunate that many of these issues fell outside the scope of this thesis and could not be discussed. The following section suggests some areas that deserve some attention and could be grounds for future research.

9.3.1. World Wide Web Privacy Issues

Issues such as the privacy implications raised by cookies, banner advertisements, and other profiling mechanisms and the effectiveness of controls such as third party filtering products could be an interesting area for exploration

9.3.2. Peer-to-Peer Security Issues

The growing popularity of peer-to-peer (P2P) networking gives rise to some important security concerns. Notable examples of peer-to-peer networking include the controversial Napster (www.napster.com) application and the Gnutella protocol and related applications (www.limewire.com; www.bearshare.com). Other current examples include Morpheus, Kazaa and Grokster.

There are a number of questions that are raised by the use of such technologies. Some of these questions include:

- Is the idea of large numbers of uncontrolled peer nodes sharing many forms of data and software fundamentally dangerous?
- Are there weaknesses in current protocols?
- How can the protocols be improved?
- Are there weaknesses in current applications?
- How can these applications be improved?
- Will peer-to-peer networking be a major source of attacks and intrusion attempts?
- How can peers be authenticated? Do we want peers to be positively identified or will peers prefer to remain anonymous?
- How will peer-to-peer change views on issues such as copyright and intellectual property?
- Will technologies such as watermarking and digital rights management be effective?
- What are the legal challenges involved?
- Will peer-to-peer have adverse effects on the performance and reliability of networks?

9.3.3. Microsoft's .Net Framework

Microsoft's .Net framework could also be an interesting area for future research. Microsoft touts this framework as being the next major paradigm in distributed systems, in some cases comparing it to the Enterprise Editions of Java 2 (Microsoft, 2002b). An examination of the security issues raised by such technologies and appropriate security measures could an interesting extension to some of the aspects covered in this thesis.

9.4. Summary

This chapter has presented the conclusions of this thesis as well as suggested some of the areas that fell outside the scope of this thesis as possible avenues for future research.

10. Appendix A: Asymmetric Encryption and Digital Signatures

10.1. Overview

Digital signature technologies use asymmetric encryption techniques in order to provide a level of trust when dealing with digital communications. As the name suggests, there are some distinct similarities between a digital signature and a handwritten signature on a physical document.

Trust is a difficult issue when dealing with an electronic medium such as the World Wide Web. It is often seen as a barrier preventing the widespread adoption of electronic commerce. Digital signatures can alleviate some of these problems as they can be used to authenticate various parties in a transaction and prove the integrity of digital documents. However, aside from their usefulness in terms of electronic commerce, digital signatures can also benefit other WWW users by providing a trust mechanism for use with executable code.

This section will begin by highlighting the importance of trust, particularly in relation to electronic commerce. However, as this thesis is primarily concerned with the risks associated with executable web content, the discussion will shift to the code signing applications of digital signature technologies.

Currently, several digital signature technologies exist, marketed and supported by a variety of vendors. This section will simply discuss the basic concepts behind digital signature technology.

10.1.1. Digital Signatures and Electronic Commerce

Security is often seen as a significant barrier restricting the widespread adoption of electronic commerce (Margherio et al., 1998) (Electronic Commerce Expert Group, 1998). Given these concerns over security, trust becomes an important issue (IBM, 1998). In an electronic environment, it can be difficult to be sure that the parties involved in a transaction are who they claim to be and transactions and communications have not been intercepted or fabricated (IBM, 1998).

Digital signature technologies aim to prove the authenticity and integrity of message or transaction (Feghhi, Feghhi & Williams, 1999, p 45). The ability to reliably assess the origin and integrity of a digital communication goes a long way towards providing a level of trust suitable for use with electronic commerce. While decisions regarding the trustworthiness of a digital message ultimately rely on human judgement, technologies such as digital signatures aim to improve our ability to make these decisions. Digital signatures are one tool to help users make informed decisions in an electronic environment (IBM 1998).

While digital signatures have the potential to play an important role in the context of electronic commerce, they can also be used to indicate trust with regard to executable program code. When used in this manner, these signatures can act as "digital shrink-wrap".

10.2. Code Signing – The "Digital Shrink-Wrap" Concept

Several commentators have used analogies comparing digitally signed program code with shrink-wrapped software purchased through retail outlets (Microsoft, 1996a) (Microsoft, 1996b) (Feghhi, Feghhi & Williams, 1999, p 99) (Garfinkel & Spafford, 1997, p169). The phrase "digital shrink-wrap" suggests similarities between signed program code and physically packaged software. When software is purchased through a retailer, there are a number of factors that indicate the authenticity of the product. Shrink-wrapping, although hardly foolproof, provides some indication that the product has not been tampered with since its release. The presence of authentic manuals and anti-piracy features such as holograms also suggest that a piece of software is authentic (Garfinkel & Spafford, 1997, p169). The appearance of the retail outlet and the reputation of the merchant can also help consumers make a decision as to the trustworthiness of the software.

When software is obtained from an electronic source such as the World Wide Web, indicators of trust are often not present or are not verifiable. Whereas in the physical world, a retail outlet may consist of bricks and mortar, the digital equivalent is often a website. Given the ease with which web sites can be created, copied and modified, it can be very difficult to establish a level of trust. Electronically obtained software usually lacks indicators such as physical manuals and anti-piracy features. There is often nothing to indicate the source of the software or anything to prove that the software has not been modified since its release (Feghhi, Feghhi & Williams, 1999, p 99). The absence of physical trust indicators necessitates other means of establishing the authenticity and integrity of a piece of software.

Code signing technologies attempt to positively identify the author of a piece of code and to prove that the code has not been tampered with since its release (Feghhi, Feghhi & Williams, 1999, p 99). As this provides similar indicators of authenticity as with physically purchased software, the term "digital shrink-wrap" is particularly apt. Additionally, if a piece of code can be shown to be malicious, positive identification of the author may make it possible for the victim to seek legal redress. Without the accountability offered by code signing technologies, publishers of a piece of malicious code may deny creating the software, or may claim that it had been modified since its release (Feghhi, Feghhi & Williams, 1999, p 100).

Code signing technologies have been enabled by the development of certain technologies and infrastructure. In order to sign code, asymmetric encryption techniques are used. In order to make this signature a meaningful way of generating trust and accountability, certificates and certificate authorities become necessary. This chapter will discuss these enabling technologies and infrastructure.

10.3. Asymmetric Encryption

Digital signatures and code signing technologies have been made possible largely because of the development of public key cryptography. The defining characteristic of this type on encryption is its use of two keys. Also referred to as asymmetric encryption, public key encryption uses different keys for encryption and decryption. Although this form of encryption requires both a public key and a secret private key, it is referred to as public key encryption rather than secret or private key encryption so as not to cause confusion with other techniques (Feghhi, Feghhi & Williams, 1999, p 36).

In order to use public key encryption, users generate two keys. One of which must be kept secret, while the other can be freely transmitted. When encrypting a message such as an email or a piece of text, a user must perform the encryption using the public key of the intended recipient. Only the holder of the corresponding private key can then decrypt the message. When used to digitally sign a document or message, the private key is used to create a signature, which can then be verified using the corresponding public key.

Simply using public key encryption to sign a digital object does not guarantee that the object is trustworthy. All that a digital signature guarantees is that the object was signed with a private key that corresponds to the public key used for verification. If the recipient of the object does not know or trust the sender, then the fact that the object is signed is effectively meaningless. Anyone could conceivably create a key pair and sign a digital object.

One solution to this problem is though the use of certificates. Certificates allow a trusted third party to vouch for the credentials of the certificate holder.

10.4. Certificates

Public key encryption itself does not guarantee that a digital object comes from a reputable source. Anyone, regardless of his or her intentions, could generate a key pair, distribute a public key and use asymmetric encryption techniques in order to gain trust. For this reason, in order to be meaningful, digital signatures usually include a certificate from a trusted third party. In effect, the trusted third party vouches for the identity of the certificate holder.

A digital certificate (or a digital ID or simply a certificate) binds information identifying an entity with a public key (Feghhi, Feghhi & Williams, 1999, p 61). Without such a binding, digital signatures are of little use and "the key is just a byte string and can be yours as well as anyone else's." (Gerck, 1998).

One common certificate format is X.509. X.509 is a standard developed by the International Telecommunication Union (ITU) (http://www.itu.int/home/) and the International Standards Organization (ISO) (http://www.iso.ch). The general structure of and X.509 certificate can be seen in Figure 5.

Certificates are issued, maintained and revoked by trusted third parties. These usually take the form of Certificate Authorities (CAs).

10.5. Certificate Authorities

Certificate Authorities (CAs) act as trusted third parties in order to vouch for the identity of various clients. Each CA is expected to publish a document describing the organisation's Certification Practice Statements (CPS).

Certificate Authorities perform a range of duties. While these duties vary between CAs there are some basic responsibilities that are common to all. Microsoft (MSDN CDROM) describes some of the duties performed by CAs as;

- They publish the criteria for granting certificates.
- They grant certificates if an applicant meets the published criteria.
- Managing certificates (enrolling, renewal, and revokation).
- Storing root keys.
- Verifying evidence submitted by applicants.
- Providing tools for enrolment.
- Accepting the liability associated with these responsibilities.

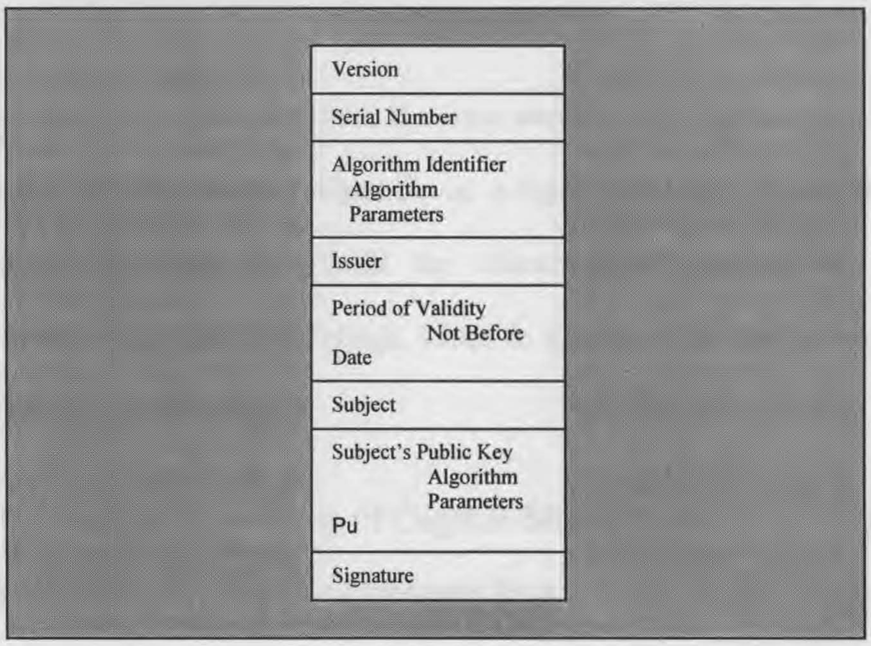


Figure 5: X.509 Certificate Structure (Microsoft Corporation, n.d)

10.6. Legal Issues and Challenges

While digital certificates and signatures provide a useful trust mechanism, they do not guarantee that a message is accurate or that a piece of code is free of malicious intent. Code signing technologies do however attempt to prove authorship of a particular piece of program code. Such proof of authorship may, in the event that a piece of program code is found to be intentionally harmful, allow victims to take legal action against the author. However given the electronic nature of these technologies and the global nature of the Internet, seeking legal redress based on digital signatures gives rise to a number of issues.

Given the current level of interest in electronic commerce it is not surprising that much has been written regarding the legal issues involved with the using digital signatures for commercial reasons. Many of the same issues apply when considering the use of digital signatures code signing purposes.

One major issue revolves around the legal standing of a digital signature compared with that of a handwritten signature on a legal document. It can be argued that electronic signatures can fulfil the characteristics required of a traditional handwritten signature (McCullagh, Little & Caelli, 1998) and as such, deserve a similar legal standing.

10.6.1. Legal Standing of Digital Signatures in Australia

In April 1998, the Electronic Commerce Expert Group (ECEG) presented a report the Commonwealth Attorney General. This report made mention of the fact that "At present the law in Australia does not generally recognise forms of electronic signatures which can perform the functions of a handwritten signature." (Electronic Commerce Expert Group, 1998). The report recommends that legislation should be put in place that deals with the legal effect of electronic signature and that other considerations should be left for the market to determine.

Many of the recommendations of the ECEG's report were based on the United Nations Commission on International Trade Law (UNCITRAL) Model Law on Electronic Commerce of 1996 (Electronic Commerce Expert Group, 1998).

The Commonwealth Government later incorporated many of the ECEG report's recommendations into the Commonwealth Government's Electronic Transactions Bill 1999.

10.7. Summary

The authenticity of digital communications, transactions and program code is often very difficult to judge. Digital signatures seek to alleviate this problem by providing the electronic equivalent of a handwritten signature. Digital signatures have been enabled largely due to the development of public key cryptography (also referred to as asymmetric encryption). Digital signatures are an important tool in improving the security of electronic communications and transactions.

Code signing is a variation of digital signature technology. It allows the author of a piece of program code to prove its origin and to prove that the code has not been modified since its release. Code signing is often described as the equivalent of digital shrink-wrap as it aims to provide users with some means to determine the trustworthiness of a piece of code.

A signature attached to a piece of code does not positively identify the author, it simply proves that the code was signed with a particular private key. Certificates are necessary to attach the identity of the author to a piece of code. These certificates are issued, maintained and revoked by certificate authorities that effectively vouch for the identity of the author.

11. Appendix B: ActiveX Development Tools

Unlike Java, ActiveX is not a language. It is a binary specification and as such any programming language or tool that can create a binary object that conforms to these stands can be used to create an ActiveX Control. However, in reality, certain languages and development tools are better suited to the development of Active Controls than others. Common development tools include Microsoft Visual C++ and Visual Basic.

Visual C++ provides a flexible, if somewhat complicated method for creating ActiveX Controls. When using a C++ environment such as Visual C++, developers have several methods for creating ActiveX Controls. Controls can be created completely by hand, or by using various class libraries or templates.

Controls can be created manually or with the help of an existing framework (Li &, Economopoulos, 1997, p.73). While creating controls manually can provide a high level of flexibility, it can be very a very tedious and inefficient method of control creation. This approach requires an in depth understanding of the inner workings of ActiveX Controls and presents a steep learning curve for developers. A much more effective way to create controls is to use an existing framework such as the Microsoft Foundation Classes (MFC), the Abstract Library Templates (ATL) or the BaseCtl framework.

The BaseCtl framework was developed by Microsoft's Visual Basic Group in 1995 to provide a framework for ActiveX Control (then referred to as OCX) development. BaseCtl was originally developed to allow the creation of small, lean controls in order to reduce the loading times for Visual Basic applications. The major disadvantage to using BaseCtl is that it is difficult to use and requires developers to implement much of the control's functionality without a great deal of help from the framework (Anderson, 1997, p. 19). BaseCtl was one of the earliest control development frameworks and has effectively been superseded by MFC and ATL.

The Microsoft Foundation Classes (MFC) are a set of C++ classes that can be used for a wide range of Windows software development projects including the creation of ActiveX Controls. The use of MFC greatly simplifies the development of controls compared with creating controls from scratch (Anderson, 1997, p. 143). The use of MFC still requires a solid understanding of the ActiveX architecture and has a considerable learning curve, although this approach is much simpler than developing controls manually or by using BaseCtl.

Given the number of developers already familiar with MFC, it seems that this would be an ideal choice for the creation of ActiveX Controls (Li & Economopoulos, 1997, p. 187). However, there is a significant drawback to using MFC for control creation and that is that controls created with MFC are often quite inefficient in terms of file size. While this may not be a significant problem in a high bandwidth intranet environment, any increase in file size can result in significant download delays across a low bandwidth network such as the Internet (Li & Economopoulos, 1997, p. 125; Anderson, 1997, p. 210). This increase in file size to due to (often unnecessary) MFC runtime code that is incorporated into the resulting control.

Microsoft's Abstract Template Libraries (ATL) provide a practical alternative to MFC for the development of ActiveX Controls. The main strength of ATL is its ability to create small, lightweight, efficient controls. Unlike MFC, ATL does not incorporate large amounts of unnecessary code into the finished control. This makes controls created with ATL well suited to the low bandwidth Internet/WWW environment. In fact, ATL has been described as a method for generating "just enough" code to implement the desired control (Li & Economopoulos, 1997, p. 21). Controls developed with ATL do not rely on specific DLLs or other libraries being included with the finished control. It seems likely that ATL will increasingly become the framework of choice for ActiveX Control development (Anderson, 1997, p. 249).

MFC enjoys a high level of integration with Microsoft's Visual C++ development environment, making it a convenient choice for the rapid development of ActiveX Controls, particularly where download times are not an important consideration (Anderson, 1997, p. 17; Li & Economopoulos, 1997, p. 20). ATL is also integrated with Visual C++, although no as tightly as MFC (Anderson, 1997, p. 18; Li & Economopoulos, 1997, p. 21). However, built-in support for ATL within Visual C++ is increasing with each new version of the language. This reflects the importance that Microsoft places in this library. BaseCtl does not have any real integration with Visual C++ (Anderson, 1997, p. 18) and is not currently considered a viable alternate for ActiveX Control development.

12. Appendix C: Internet Explorer Zones

The following table summarises the differences between the pre-defined security levels for Microsoft's Internet Explorer Web browser. Each of the four security zones used by the browser can be configured to use either the High, Medium, Medium-Low or low security profile listed in the table below. Additionally the browser allows users to customise these profiles.

	Security Level			
Property	High	Medium	Medium-Low	Low
Download signed ActiveX controls	Disable	Prompt	Prompt	Enable
Download unsigned ActiveX controls	Disable	Disable	Disable	Prompt
Initialise and script ActiveX controls not marked as safe	Disable	Disable	Disable	Prompt
Run ActiveX controls and Pug-ins	Disable	Enable	Enable	Enable
Script ActiveX controls marked as safe	Enable	Enable	Enable	Enable
Allow Cookies	Disable	Enable	Enable	Enable
Allow per-session cookies	Disable	Enable	Enable	Enable
File download	Disable	Enable	Enable	Enable
Font download	Prompt	Enable	Enable	Enable
Java Permissions	High Safety	High Safety	Medium Safety	Low Safety
Access data sources across domains	Disable	Disable	Prompt	Enable
Drag and drop or copy and paste files	Prompt	Enable	Enable	Enable
Installation of desktop items	Disable	Prompt	Prompt	Enable
Launching programs in an IFRAME	Disable	Prompt	Prompt	Enable
Navigate Sub frames across different domains	Disable	Enable	Enable	Enable
Software channel permissions	High Safety	Medium Safety	Medium Safety	Low Safety
Submit non-encrypted form data	Prompt	Enable	Enable	Enable
User data persistence	Disable	Enable	Enable	Enable
Active scripting	Enable	Enable	Enable	Enable
Allow paste operations via script	Disable	Enable	Enable	Enable
Scripting of Java Applets	Disable	Enable	Enable	Enable
Logon	Prompt for username and password	Automatic logon only in Intranet zone	Automatic logon only in Intranet zone	Automatic logon with current username and password

13. Appendix D: Windows NT/2000/XP Security Architecture

This appendix provides information regarding the security architecture of the Windows NT line of operating systems. This line also includes Windows 200 and Windows XP.

Common operating systems, particularly those for WWW clients and servers include Windows 95/98, Windows NT Server and Workstation, UNIX, Linux and MacOS. In terms of security features, these operating systems vary greatly. While Unix and Windows NT offer some important security mechanisms, the security features of Windows 95/98 are considered minimal.

13.1. Background

Microsoft's Windows Operating System is currently the world's most prolific desktop operating system (add reference here). However, Windows is not one single product. Rather the name represents a family of operating systems. Currently the Windows family contains a number of product lines, primarily Windows 3.1,Windows 95/98, Windows NT/2000 and Windows CE.

This thesis will refer to the Windows 95/98/ME line of Microsoft of operating systems as Windows9X or Win9X. Architecturally, these operating systems are quite similar and as such, they will be discussed as if they are essentially one product. As this thesis focuses on Windows NT and Windows 2000 in detail, it will refer to products individually despite the fact that there are a number of architectural similarities.

At a superficial level, there are some distinct similarities between the different branches of the Windows family. Windows 3.1 and Windows NT 3.5 share a similar user interface, as do Windows 95/98 and Windows NT 4.0. Despite these similarities, the different Windows product lines were developed with under different circumstances and with different goals. As a result, there are many important architectural differences between Windows NT and other member of the Windows family.

13.2. Characteristics of Windows NT/ 2000

While there are a number of cosmetic similarities between the Win9x and Windows NT/2000 product lines, there a also a number of important architectural differences. This is not surprising as both product lines are aimed at different segments of the Operating System market. Win9x is generally a consumer level operating systems aimed at home users. In contrast Windows NT and Windows 2000 are aimed at a number of market segments. There are several variations of Windows NT and Windows 2000. There are variations aimed at professional users, designed for desktop workstations, as well as several variations designed for use as servers. As this thesis focuses on security threats faced by consumers of World Wide Web services, it will discuss only the "Professional" versions of Windows NT and Windows 2000. Others variations of these operating systems, while architecturally similar, fall outside the scope of this thesis.

In contrast to the Windows 9x line, Windows NT and 2000 were designed to be quite portable. Whereas Windows 9x is limited to Intel based platforms, Windows NT/2000 versions have been released for other platforms. However Intel remains a popular choice of platform for this operating system. Unlike Win9x, Windows NT and 2000 make use of a Hardware Abstraction Layer (HAL) in order to insulate most of the Operating System from hardware dependencies introduced by various platforms. This HAL can be seen in Figure 6.

While much of Windows9x was written in platform dependent assembly code, NT and 2000 were developed using higher-level languages. The use of higher-level languages and the inclusion for the HAL makes Windows NT and 2000 much easier to port to platforms other than those based on Intel processors.

While Windows 3.1 and Win9x were heavily dependent on the MS-DOS operating system, Windows NT and 2000 are completely independent of this earlier operating system. Unlike Windows 3.1, Windows NT and 2000 do not rely on having MS-DOS installed and unlike the Win9x line, Windows NT and 2000 do not incorporate large portions of MS-DOS technology. As such Windows NT and 2000 differ greatly in terms of architecture when compared with Win9x.

Robustness, stability and security were also major design goals of Windows NT and 2000. Whereas the security features implemented by Win9x can only be described as minimal, Windows NT and 2000 do implement some important security features (Sheldon, 1997, p 76; Rutstein, 1997, p3), many of which will be discussed in this chapter

13.3. The Windows NT Architecture

Architecturally, WinNT is very different to the Win9x line of operating systems. It is divided into several distinct subsystems and components. The basic architecture of the Windows NT can be seen in Figure 6.

One notable architectural feature is that Windows NT draws a clear distinction between User Mode and Kernel Mode. All user applications execute in User mode while various system components execute in kernel mode. The intention behind this division is to ensure that the kernel remains intact and running even if indivi ıal applications prove to be unstable. As a result, unstable applications should not affect the stability of the whole operating system.

As shown in Figure 6, Windows NT is capable not only of running Win32 based but also some OS/2 and POSIX applications. Each of these types of applications is executed via the appropriate subsystem, each of which is executed in user mode. Figure 6 also shows that some sections of the security subsystem are executed in user mode while the Security Reference Monitor (discussed later in this chapter) executes in kernel mode.

Figure 6 also shows the Hardware Abstraction layer (HAL) and its relationship with other subsystems. As stated previously, this layer insulates much of the Windows NT Operating System from hardware specific dependencies.

13.4. The Windows NT Security Architecture

Windows NT offers a range of security features that are not available in many consumer desktop operating systems such as Win9x. The Windows NT security architecture is based on three key components - the Local Security Authority (LSA), Security Account Manager (SAM) and the Security Reference Monitor (SRM). These components are described in depth by a number of authors (Kelley, Mayson, 1997; Sheldon, 1997) and their relationship can be seen in Figure 6.

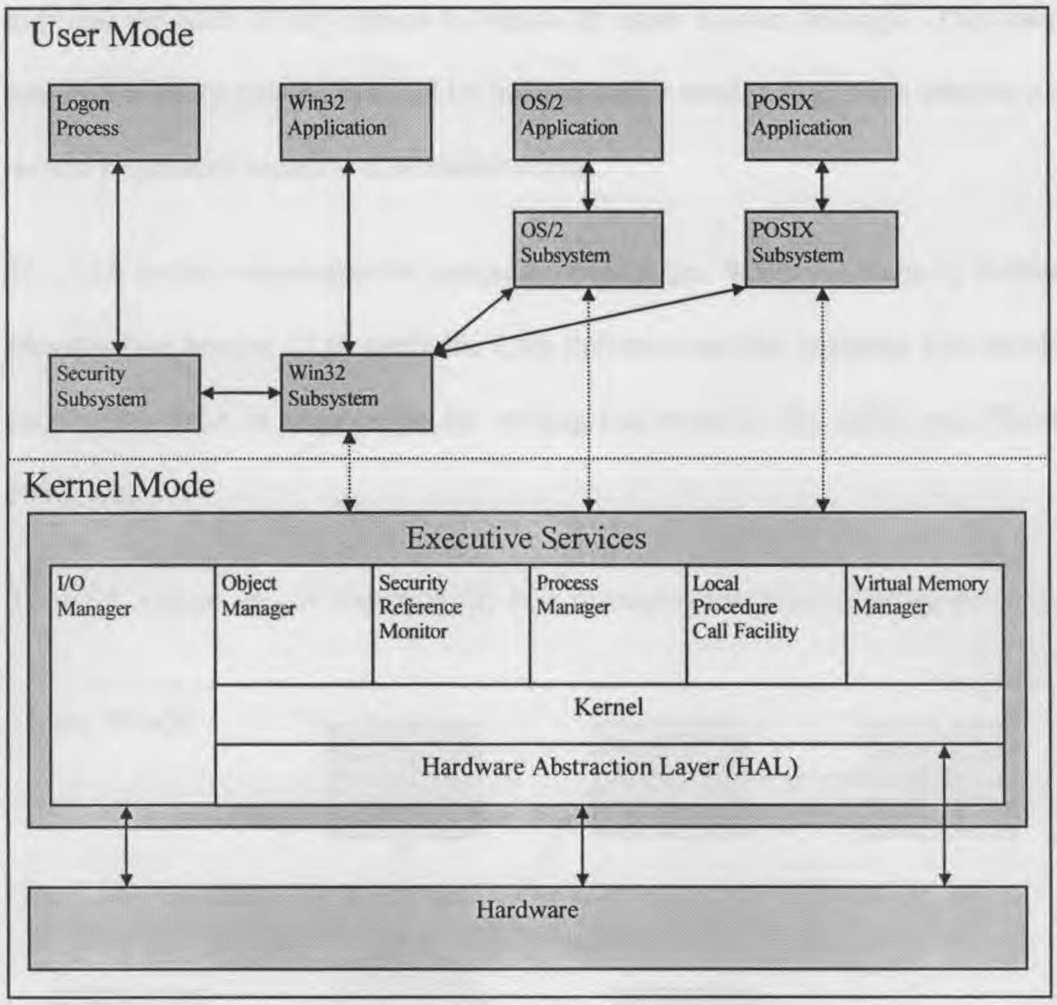


Figure 6: Windows NT/2000 Architecture

13.5. The Local Security Authority and Logon Process

The heart of the Window NT Security architecture is the Local Security Authority (LSA), as can be seen in Figure 7. The LSA is responsible for generating access tokens, managing security policies and controlling the auditing process (Rutstein, 1997, p. 8).

The Logon Process allows both local and remote users to logon to a Windows NT machine. Once users are successfully logged on, they are identified by a Security Identifier (SID) and an Access Token. The LSA is responsible for generating access tokens as users complete the logon process. This token incorporates the SID of the user and the SIDs of any groups to which the users account belongs. This token is attached to every process invoked by the user and is used to determine whether a user should be granted access to a particular object.

The LSA is also responsible for managing audit logs. When the Security Reference Monitor (see section 13.6) alerts the LSA that an event has occurred that should be audited, the LSA is responsible for writing that event to the audit logs (Rutstein, 1997, p 8).

The LSA's other area of responsibility is in managing the security policy database.

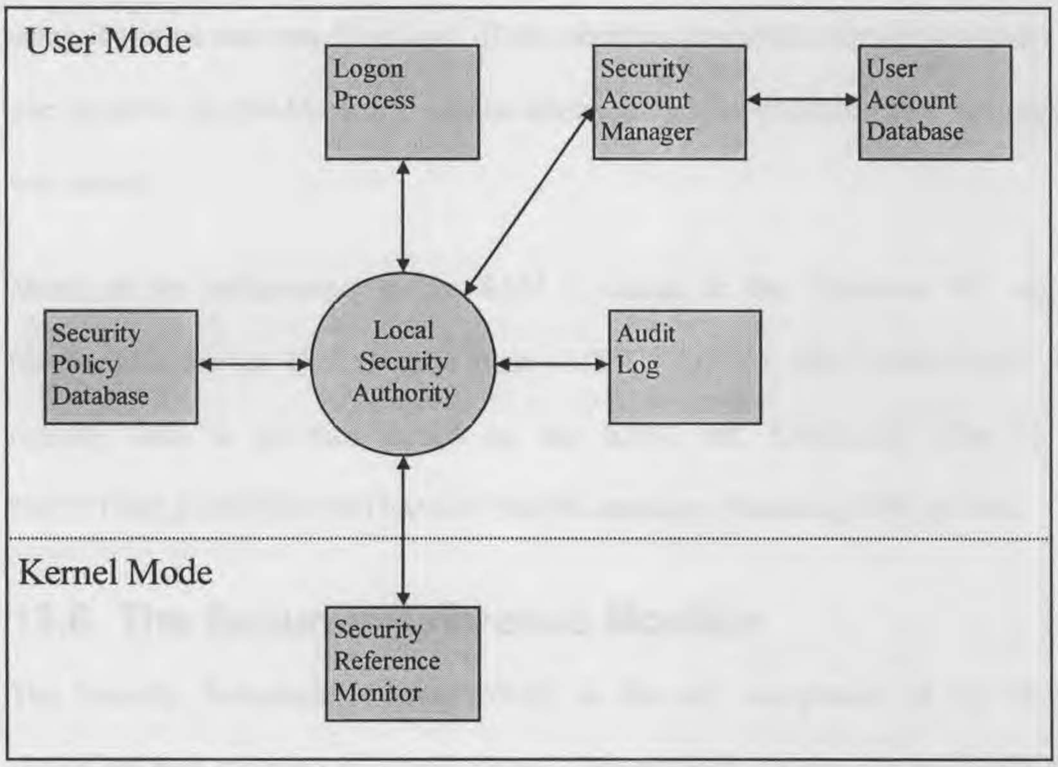


Figure 7: Windows NT Security Architecture

The Security Account Manager (SAM) controls a database of account information. This database contains information regarding user and group accounts. During the logon process, the SAM consults the User Account Database and returns the user's SID to the LSA (Kelley & Mayson, 1997).

The SAM contains information about user accounts including passwords. In most cases, Windows NT stores two passwords - a native Windows NT password and a password for backward compatibility with Microsoft's LAN Manager product line. Both passwords are encrypted twice using one-way functions before being stored in the SAM. As one-way functions are used for the two encryption processes, it is technically very difficult for a plaintext password to be retrieved from its encrypted form. When password checking is performed, the password to be tested is encrypted using the same one-way functions. If the resulting encrypted password matches the one stored in the SAM, then it can be assumed that the password that was entered was correct.

Much of the information in the SAM is stored in the Windows NT registry (discussed in section 13.7.3) under the key HKEY_LOCAL_MACHINE\SAM. This registry data is in turn stored in the SAM and SAM.LOG files in the %SYSTEM_ROOT%\SYSTEM32\CONFIG directory (Rutstein, 1997, p 144).

13.6. The Security Reference Monitor

The Security Reference Monitor (SRM) is the only component of the security subsystem that executes in kernel mode. This module is primarily responsible for comparing an access token (as generated by the LSA) against the permissions set for an object and determining the level of access granted, if any.

Objects such as files, threads and registry keys all have an attached Security Descriptor (SD). This SD includes, among other attributes, the SID of the object's owner, an Access Control List (ACL) and a system ACL (Sheldon, 1997, p. 86).

In most cases, the owner of an object is the user that first created the object. However, in some cases it is possible for other user to take owner ship of a file.

ACLs are central to Windows NT's object security architecture. ACLs contain zero or more Access Control Entries (ACEs). Each ACE specifies a SID representing a user or group and a set of permissions assigned to that SID. The SRM is responsible for checking a user's access token against the entries in an ACL in order to determine whether the level of access requested by the user should be granted. The SRM scans through all of the entries in an ACL and accumulates any access permissions assigned to the user until the permissions granted match the permissions requested or the SRM reaches the end of the ACL. In the former case the SRM would grant the user the desired access while in the latter case, access would be denied (Sheldon, 1997, p. 87). It is possible that an ACE will specifically deny all access to a particular group or user, in which case this ACE will revoke any permissions granted by other ACEs in the object's ACL (Sheldon, 1997, p. 88). In effect, access to an object is denied unless an ACE specifically grants permission. Additionally access is denied if any ACE denies the user the requested permissions even if another ACE specifically grants the user these permissions (Rutstein, 1997, p. 12).

When the SRM makes the decision to grant or deny access to an object, it generates the necessary audit event notification messages and passes them to the LSA, which in turn adds entries into various audit logs.

13.7. Securing Windows NT

Despite the security features offered by Windows NT, default installations are quite relaxed in terms of security. Administrators must be careful to properly configure NT to make full use of its security features.

When attempting to secure an installation of Windows NT, there are several areas that need to be considered including users and groups, file systems, registry access, network configuration, services packs, updates and hotfixes.

13.7.1. Managing User and Group Accounts

Windows NT allows for the creation of user and group accounts. In most cases, users will have their own individual accounts. These accounts can belong to one or more groups and each group can contain any number of user accounts. Groups can simplify the process of assigning access rights and privileges to users. Instead of assigning rights and privileges to each individual account, they can be assigned to groups and then inherited by the members of these groups.

Administrators can create, modify and delete accounts and groups. In addition, Windows NT provides several in-built groups and accounts some of which deserve special attention as they have significant implications for the security of Windows NT systems. These include the Everyone group and the Administrator and Guest Accounts.

The Everyone Group includes every user that accesses a Windows NT System (Rutstein, 1997, p37). In fact it is impossible to create a user account that is not a member of the Everyone group. By default, Windows NT assigns the Everyone group several permissions including; full control over new file shares, the ability to change permissions on the root directories of any NTFS partition and the ability to change the permissions of the System32 directory (Sheldon, 1997, p. 181). This is one example of how Windows NT security relies on careful configuration by administrators. The default security settings in this area do not lend themselves to the creation of a secure environment straight "out of the box". While in most cases it is a simple task for administrators to take such permissions away from the Everyone group, this issue illustrates the importance of proper configuration rather than relying on default security settings.

There are also certain issues surrounding the use of the administrator and guest accounts. Due to the powerful nature of the Administrator account, it is recommended that administrators create their own personal accounts for day-to-day work such as word processing and Internet access. In contrast, the administrator account should only be used for administrative duties. Given the permissions associated with the Administrator account, any malicious code executed by some using the administrator would have largely unrestricted access to a Windows NT system. This is not a new concept, nor is it restricted to Windows NT. It is considered good practice when using any operating system that allows different users to be awarded different levels of permissions to not use highly privileged accounts for mundane duties.

The in-built Guest account also deserves special consideration. The guest account allows users without and existing account to logon on to a Windows NT machine, albeit with very minimal permissions. In most cases, users will not even need a password in order to logon as a guest. The Guest account is a member of the Guests group and also the Everyone system group. As the Guest account is member of the Everyone group, by default it will have access to shared directories, unless permissions for the Everyone group are specifically revoked. In addition, any number of users may share one guest account and as such, audit logs will not reveal any information about any particular guest account user (Sheldon, 1997, p. 98). In versions prior to Windows NT 4.0, the guest account was enabled by default. This was changed with the release of Windows NT 4.0. The guest account is now disabled by default and if needed, must be specifically re-enabled.

13.7.2. File System Security

Windows NT variants prior to version 4.0 allowed the use of three distinct file systems - File Allocation Table (FAT), High Performance File System (HPFS) and New Technology File System (NTFS). Windows NT 4.0 only supports the use of FAT and NTFS, and as such, this section will only discuss these two file systems.

FAT is the file system made popular by the MS-DOS operating system and Windows 95/98. While it can be used with Windows NT, it offers no advantages in terms of security and will not be discussed here in depth. NTFS is the "Native" file system of Windows NT and offers several security advantages over other file systems.

The advantages of using the NTFS file system include speed improvements, reduced file fragmentation, small cluster sizes to reduce waste, file and directory compression (Kelley, Mayson, 1997). In addition, NTFS is the only file system that allows administrators to make use of the file permission mechanisms offered by Windows NT (Rutstein, 1997, p. 66).

Having the choice of two file systems also raises several issues. As the FAT file system can be used by both Windows 95/98 and Windows NT, there is a risk that FAT volumes my be accessed by operating systems other than Windows NT. Even though file permissions cannot be set on FAT volumes, in most cases users will still need to log Windows NT using a defined account in order to access files. However, this can easily be bypassed by installing another operating system on the machine such as Windows 95/98 or MS-DOS.

As NTFS cannot be used by Windows 95/98, any volumes formatted with this file system will be invisible to users not using Windows NT. While it is true to say that Windows 95/98 cannot access NTFS volumes natively, utilities such NTFSDOS can give operating systems such as Windows 95/98 read-only access to any NTFS volumes on a particular machine. This has significant implications for Windows NT file system security as security mechanisms such as the logon process and file system permissions can be bypassed simply by booting a machine with an operating system such as Windows 95/98 or MS-DOS, running NTFSDos and copying sensitive files over to FAT based hard disk or removable media. As Windows NT is not even running, the Security Reference Monitor cannot prevent access to such sensitive files and the LSA cannot audit the file access. Third party encryption tools can provide some protection against this type of situation. Physically securing Windows NT machines must also be an important consideration.

Unlike the FAT file system, NTFS allows files to be owned by particular users or groups. It also allows the setting of access permissions on files and directories and offers provisions for auditing file accesses.

File system security is quite relaxed in a default installation of Windows NT, particularly for the Workstation version. It is the system administrator's responsibility to ensure that, where required, file system security features are used. When created, files give access to the Everyone group and as a result any other user would have access to this file. It is the responsibility of the creator of the file and system administrators to ensure that appropriate access restrictions are placed on the file

However, file system security can be a complex issue. While file ownership and access may be quite straightforward when dealing with user's documents such as word processing and spreadsheet files, it can be difficult to assign permissions to certain system files. Not specifying permissions on such files may have certain security implications, while restricting access too tightly may interfere with the normal operation of a particular system. Some authors including Sheldon (1997) describe some of the default permissions on key system files and make some suggestions as to how permissions may be set safely.

13.7.3. Registry Security

Both the Windows 95/98 and Windows NT product lines make use of a centralised database to store various users, hardware and configuration settings. Not only does it govern the behaviour of hardware and application software, but also the operating system itself. As a result, measures should be taken to guard against accidental or deliberate tampering.

The Windows NT registry is a structured hierarchy of hives, keys, sub-keys, values and data and is similar in structure, but not identical to the Windows 95/98 registry. The top level of the Windows NT registry is divided into five major groupings referred to as hives. These hives can be seen in Table 2. These hives are in turn divided into a number of keys and sub-keys.

Hive	Description
HKEY_CLASSES_ROOT	Contains information about registered software components including COM/OLE and ActiveX Controls.
HKEY_CURRENT_USER	Contains information regarding the user that is currently logged on.
HKEY_LOCAL_MACHINE	Contains information regarding the local Windows NT machine. It includes information about drivers, installed hardware and software, system

	configuration and security settings.
HKEY_USERS	Contains information about all users of the local machine.
HKEY_CURRENT_CONFIG	Contains information about the current configuration of the local machine.

Table 2: Windows NT Regsitry Structure (Rutstein, 1997, p. 143)

Given the wealth of information stored in the Windows NT registry, it should be obvious that some measures will be need to prevent accidental or malicious modification of registry information (Rutstein, 1997, p144). To further complicate matters, users may attempt to modify registry settings remotely on any Windows NT machine on which the user has an account. User may also try to connect to a machine using a guest account in order to perform remote registry modifications (Rutstein, 1997, p144).

The Windows NT registry can be secured in a similar manner to an NTFS file system volume. Permissions can be added to keys in much the same way as they can be added to files and directories. However the same difficulties remain. As with file system permissions, it can be very difficult to determine the level of permissions that should be assigned to certain keys. Some keys are relied upon by the Windows NT System and/or user application. Placing tight restrictions on these keys may prevent the system or applications from performing properly. In contrast, lax permissions may adversely affect the security of a system. The sheer number of keys in the registry and the importance of this database to a Windows NT system can make the setting of permissions a difficult task. A number of authors present guidelines suggesting permissions that can be applied to certain registry keys (Sheldon, 1997, p.627; Rutstein, 1997, p. 148; Jumes, Cooper, Chamoun & Feinman, 1999, p. 191).

In addition to setting registry permissions, access to registry keys can also be audited. However given the sheer number of keys in the Windows NT registry, the addition of auditing information can greatly increase the overall size of the registry. In addition, performance overheads involved with auditing may be significant if a large number of keys are to be audited.

13.7.4. Network Security

As this thesis is mainly concerned with security issues that affect Windows NT platforms as Internet and World Wide Web clients, Windows Networking security will not be discussed here in detail.

13.7.5. Service Packs, Patches and Hotfixes

Since the release of Windows NT 4.0 in 1996, Microsoft has released a number of official updates in the form of service packs, patches and Hotfixes. It is important for administrators to be aware of the latest official updates and the issues that they address. Currently six service packs have been released for Windows NT 4.0.

13.8. Summary

Microsoft Windows NT and 2000 are members of a larger family of Microsoft operating systems. This thesis focuses on the Windows NT/2000 variants that are designed to be used on workstations and desktop machines. These operating systems were chosen due to the fact that they implement a number of important security features and are commonly used on WWW client machines.

This chapter describes the security features and architecture of Windows NT/2000. Later chapters will build on this discussion and argue the advantages and disadvantages of operating system level controls in terms of reducing risks posed by WWW usage.

14. List of References

Anderson, J. (1997). ActiveX Programming with Visual C++ 5.0. Que Corporation.

Appel, A. (1999) Protection against untrusted code: The JIT compiler security hole, and what you can do about it. [On-line]. Available WWW: http://www-4.ibm.com/software/developer/library/untrustsed-code. [16/11/1999]

BrowserNews. (2002). Browser News: Statistics. [On-line]. Available WWW: http://www.upsdell.com/BrowserNews/stat.htm [29/10/2002]

CERT. (1997). Security of the Internet. [On-line]. Available WWW: http://www.cert.org/encyc_article/tocencyc.html [13/09/2002]

CERT. (1998). CERT Vulnerability Note VN-98.07. [On-line]. Available WWW: http://www.cert.org/vul_notes/VN-98.07.backorifice.html [13/09/2002]

CERT (1999a). Frequently Asked Questions About the Melissa Virus. [On-line]. Available WWW: http://www.cert.org/tech_tips/Melissa_FAQ.html [13/09/2002]

CERT. (1999b). Vulnerability Note VU#24839. [On-line]. http://www.kb.cert.org/vuls/id/24839 [03/06/2002]

CERT. (2000a). CERT® Advisory CA-1999-04. [On-line]. http://www.cert.org/advisories/CA-1999-04.html [28/07/2002]

CERT. (2000b). CERT® Advisory CA-2000-04 Love Letter Worm. [On-line]. http://www.cert.org/advisories/CA-2000-04.html [06/08/2002]

CERT. (2000c). Results of the Security in ActiveX Workshop. [On-line]. Available WWW: http://www.cert.org/reports/activeX_report.pdf. [09/02/2001].

CERT. (2001). Vulnerability Note VU#320944. [On-line]. http://www.kb.cert.org/vuls/id/3209441 [03/06/2002]

Chess, D., Morar, J. (1998). Is Java Still Secure. [On-line]. Available WWW: http://www.research.ibm.com/antivirus/SciPapers/Morar/JavaSecure.html. [12/11/2002]

CIAC. (2001). L-062: Erroneous Verisign-Issued Digital Certificates for Microsoft. [On-line]. http://www.ciac.org/ciac/bulletins/l-062.shtml [06/08/2002]

Cohen, F. (1984). Computer Viruses - Theory and Experiments. [On-line]. Available WWW: http://www.all.net/books/virus/ [13/09/2002]

Edwards, M. (1997). Lets, talk about Java Portability. [On-line]. Available WWW: http:// http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnwebtool/html/msdn_javaport.asp [02/04/2003]

Electronic Commerce Expert Group. (1998). Electronic Commerce: Building The Legal Framework. [On-line]. Available WWW: http://www.law.gov.au/aghome/advisory/eceg/ecegreport.html. [11/11/2002].

Feghhi, J., Feghhi, J., Williams, P. (1999). Digital Certificates. Applied Internet Security. Addison Wesley Longman.

Garfinkel, S., Spafford, G. (1997). Web Security & Commerce. O'Reilly and Associates

Gerck, E. (1998). Overview of Certification Systems. [On-line]. Available WWW: http://www.mcg.org.br/cert.htm. [11/11/2002]

Gong, L., Mueller, M., Prafullchandra, H., Schemers, R. (1997). Going Beyond the Sandbox: An Overview of the New Security Architecture in the Java Development Kit 1.2. [On-line]. Available WWW: http://java.sun.com/people/gong/papers/jdk12arch.ps.gz [10/09/2002]

Gong, L. (1998). Java Security Architecture (JDK1.2). [On-line]. Available WWW: ftp://ftp.javasoft.com/docs/jdk1.2/security-spec.pdf. [7/12/1999]

Gosling, J., McGilton, H. (1996) The Java Language Environment A White Paper. [On-line]. http://java.sun.com/docs/white/langenv/index.html [12/11/2002]

Hamilton, G. (2001). Java Beans Component APIs for Java. [On-line]. Available WWW: http://java.sun.com/javaone/javaone96/pres/PlatInd.pdf [13/09/2002]

Henry, D.,Cook, S., Buckley, P., Dumagan, J., Gurmukh, G., Pastore, D., LaPorte S. (1999). The Emerging Digital Economy II. [On-line]. Available WWW: http://www.esa.doc.gov/508/esa/TheEmergingDigitalEconomyII.htm [11/11/2002]

Hopwood. (1997). *A Comparison between Java and ActiveX Security.* . [On-line]. http://www.users.zetnet.co.uk/hopwood/papers/compsec97.html [20/08/2002]

IBM. (1998). Securing IBM Applications with Publick Key Infrastructure. [On-line]. Available WWW: http://www-3.ibm.com/security/library/wp_pki0730.shtml [11/11/2002]

JDK 1.0 [Computer Software]. (1995) [on-line]. Available WWW: http://java.sun.com/products/jdk/1.0.2/

JDK 1.1 [Computer Software]. (1996) [on-line]. Available WWW: http://java.sun.com/products/jdk/1.1/

JDK 1.2 [Computer Software]. (1998) [on-line]. Available WWW: http://java.sun.com/products/jdk/1.2/

JRE [Computer Software]. (1998) [on-line]. Available WWW: Http://www.javasoft.com

Jumes, J., Cooper, N., Chamoun, P., Feinman, T. (1999). Microsoft Windows NT 4.0 Security, Audit and Control. Microsoft Press.

Koved, L., Nadalin, A., Neal, Don., Lawson, T. (1998) The Evolution of Java Security. [On-line]. Available WWW: http://www.research.ibm.com/journal/sj/373/koved.html. [11/11/2002]

LaDue, M. (n.d). A Collection of Increasingly Hostile Applets. [on-line]. Available WWW: http://www.cigital.com/hostile-applets/ [24/09/2002].

Li, S., Economopoulos, P. (1997). ActiveX / COM Control Programming. Birmingham: Wrox Press Ltd.

Lock, A. (2002). Mozilla ActiveX Project. [on-line]. Available WWW: http://www.iol.ie/~locka/mozilla/mozilla.htm [24/09/2002]

Margherio, L., Henry, D., Cooke, S., Montes, S., Hughes, K. (1998). The Emerging Digital Economy. [On-line]. Available WWW: http://www.esa.doc.gov/508/esa/pdf/EmergingDig.pdf [02/10/2002]

McCullagh, A., Little, P., Caelli, W. (1998). Electronic Signatures: Understand the Past to Develop the Future. University of NSW Law Journal. 21(2). 452-466.

McGraw, G., Felten, E. (1998). New Issues in Java Security: How the sandbox simultaneously evolved into JDK 1.2 and devolved into Card Java. [On-line]. Available WWW: http://www.rstcorp.com/javasecurity/compstrat.html

McManis, C. (1996). The basics of Java class loaders . [On-line]. Available WWW: http://www.javaworld.com/javaworld/jw-10-1996/jw-10-indepth-p2.html [12/11/2002]

McMullin, B. (2000). John von Neumann and the Evolutionary Growth of Complexity: Looking Backward, Looking Forward.... [On-line]. Available WWW: http://www.eeng.dcu.ie/~alife/talks/alife7/vn-complexity/html-single/ [13/09/2002]

Microsoft Corporation. (n.d). Authenticode Appendixes. [On-line]. Available WWW: http://msdn.microsoft.com/workshop/security/authcode/appendixes.asp [12/11/2002]

Microsoft Corporation. (1997). Microsoft, Sun and Java. [On-line]. Available WWW: http://www.microsoft.com/presspass/java/default.asp [12/11/2002]

Microsoft Corporation, (1999a). Chapter 6 – Digital Certificates. [MS TechNet CD].

Microsoft Corporation, (1999b). Chapter 7 – Security Zones and Permission-Based Security for MS Virtual Machine. [MS TechNet CD].

Microsoft Corporation, (1999). Info: Difference Between OLE Controls and ActiveX. [On-line]. Available WWW: http://support.microsoft.com/support/kb/articles/Q159/6/21.asp. [10/11/1999]

Microsoft (2002a). Information on the VBS/Loveletter Virus. [On-line]. Available WWW: http://www.microsoft.com/technet/treeview/default.asp?url=/TechNet/security/virus/vbslvltr.asp [13/09/2002]

Microsoft (2002b). Microsoft .NET Pet Shop 2.0. [On-line]. Available WWW: http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnbda/html/bdasamppet.asp [13/11/2002]

Nachenberg, C. (n.d). Computer Parasitology. [On-line]. Available http://enterprisesecurity.symantec.com/pdf/computerparasitology.pdf?PID=na&EID=2 [29/07/2002]

Naviscope [Computer Software]. (2001) [on-line]. Available WWW: http://www.naviscope.com/

NUA Internet Surveys. (2002). OneStat: New browsers take on Internet Explorer. [On-line]. Available WWW: http://www.nua.ie/surveys/index.cgi?f=VS&art_id=905358103&rel=true [25/09/2002]

Oaks, S. (1998a). Java Security: Chapter 3. Java Class Loaders. [On-line]. Available WWW: http://octopus.cdut.edu.cn/~yf17/javaent/security/ch03_01.htm [12/11/2002]

Oaks, S. (1998b). Java Security: Chapter 4. The Security Manager Class. [On-line]. Available WWW: http://octopus.cdut.edu.cn/~yf17/javaent/security/ch04_01.htm [12/11/2002]

Pfleeger, C (2000). Security in Computing. New Jersey: Prentice Hall, Inc.

Pipkin, D. (2000). Information Security. . New Jersey: Prentice Hall, Inc.

Pistoia, M., Relle, D., Gupta D., Nagnu, M., Raman, A. (1999). Java 2 Network Security. [On-line]. Available WWW: http://www.redbooks.ibm.com/abstracts/sg242109.html [10/09/2002]

Rutstein, C. (1997). National Computer Security Association Guide to Windows NT Security. McGraw-Hill

Schneier, B. (2000). Secrets and Lies. John Wiley & Sons, Inc: New York.

Schneier, B. (2002a). Cryptogram Newsletter. [On-line]. Available http://www.counterpane.com/crypto-gram-0202.html#1 [13/05/2002]

Schneier, B. (2002b). Cryptogram Newsletter. [On-line]. Available http://www.counterpane.com/crypto-gram-0101.html#10 [13/05/2002]

Schneier, B. (2002b). Cryptogram Newsletter. [On-line]. Available http://www.counterpane.com/crypto-gram-0101.html#10 [13/05/2002]

Skrenta, R. (n.d). Elk Cloner (circa 1982). [On-line]. Available WWW: http://www.skrenta.com/cloner/ [13/09/2002]

Sheldon, T. (1997). Windows NT Security Handbook. McGraw-Hill.

Sophos Anti-Virus (2002). Melissa worm author sentenced to 20 months. [On-line]. Available WWW: http://www.sophos.com/pressoffice/pressrel/uk/20020501smith.html [13/09/2002]

Sun Micrososystems. (1996). The Java Language - An Overview. [On-line]. Available WWW: http://java.sun.com/docs/overviews/java/java-overview-1.html [13/09/2002]

Sun Micrososystems. (1999). 100% Pure Java Certification Program. [On-line]. Available WWW: http://java.sun.com/100percent/ [04/02/2000].

Sun Micrososystems. (2000). Products & APIs. [On-line]. Available WWW: http://www.javasoft.com/products/

Sun Microsystems. (2002) Chronology of security-related bugs and issues, 3/19/02. [On-line]. Available http://java.sun.com/sfaq/chronology.html [20/08/02]

Sun Microsystems, (n.d). Sun Microsystems Takes Legal Action Against Microsoft. [On-line]. Available http://www.sun.com/announcement/letter.html [12/11/2002]

Venners, W. (2002) Security and the Class Loader Architecture. [On-line]. Available: http://www.artima.com/underthehood/classloaders.html [12/11/2002]

WebWasher [Computer Software]. (2002) [on-line]. Available WWW: http://www.webwasher.com

White, S., Kephart, J., Chess, D. (1995). Computer Viruses: A Global Perspective. [On-line]. Available WWW: http://researchweb.watson.ibm.com/antivirus/SciPapers/White/VB95/vb95.distrib.ht ml [13/09/2002]

Wong, W. (1998). Sun vs. Microsoft: Political Battle Over Java. [On-line]. Available WWW: http://www.techweb.com/wire/story/TWB19981106S0002. [04/02/2000]

Bibliography

Austalian Bureau of Statistics. (2000). Communication and Information Technology. Use of Inormation Technology. [On-line]. Available WWW: http://www.abs.gov.au/websitedbs/c311215.NSF /Australia+Now+- +A+Statistical+Profile /09C60548FF693D4FCA256863001C1FFD[16/11/1999].

Black, U. (1994). TCP/IP and Related Protocols. McGraw-Hill.

Berners-Lee, T. (1999). About The World Wide Web Consortium. [On-line]. Available WWW: http://www.w3.org/Consortium/ [08/02/2000]

Caelli, W., Longley, D., Shain, M. (1994). Information Security Handbook. Macmillan Press Ltd.

Cheswick, W.R., Bellovin S.M. (1994). Firewalls and Internet Security. Addison-Wesley Publishing Company.

Cohen, F.B. (1995). Protection and Security on the Information Superhighway. New York: John Wiley & Sons, Inc.

Dean, D., Felton, E.W., Wallach, D.S, (1996). Java Security: From HotJava to Netscape and Beyond. [On-line]. Available WWW: http://www.cs.princeton.edu/sip/pub/secure96.html

Dietl, J. (1998). World Wide Web Consortium [W3C] Backgrounder. [On-line]. Available WWW: http://www.w3.org/Press/Backgrounder.html. [08/02/2000].

Electronic Frontiers Australia. (1999). Campaign against Australian Internet Censorship Legislation. [On-line]. Available WWW: http://www.efa.org.au/Campaigns/stop.html

Electronic Frontiers Australia. (2000). On-Line Privacy Issues. [On-line]. Available WWW: http://www.efa.org.au/Issues/Privacy/Welcome.html#bill. [08/12/2000].

Felten, E. (1999). SIP: News. [On-line]. Available WWW: http://www.cs.princeton.edu/sip/history/index.php3. [11/11/2002]

Gibson, S. (2001a). OptOut – Aureate Spyware. [On-line]. Available WWW: http://grc.com/oo/aureate.htm. Downloaded 11/04/2001.

Gibson, S. (2001b). The Anatomy of File Download Spyware . [On-line]. Available WWW: http://grc.com/downloaders.htm. Downloaded 11/04/2001.

Fites, P., Kratz, M.P.J (1993). Information Systems Security A Practitioners Approach. Van Nostrand Rheinhold.

Gordon, S., Chess, D. (1998). Where There's Smoke There's Mirrors: The Truth About Trojan Horses on the Internet. [On-line]. Available WWW: http://www.av.ibm.com/InsideTheLab/ScientificPapers/ Gordon/Trojan/html

Hamilton, G. (1997). JavaBeans. [On-line]. Available WWW: http://www.javasoft.com/beans/docs/ spec.html[07/06/1999]

Howard, J. (1997). An Analysis Of Security Incidents On The Internet 1989 – 1995. [On-line]. Available WWW: http://www.cert.org/research/JHThesis/Word6/. Downloaded 10/02/2000.

Kindel, C. (1997). ActiveX and The Web - Architecture & Technical Overview. [On-line]. Available WWW: http://www.microsoft.com/com/presentations/default.asp

Lalonde, G, (2001). The Spyware Infested Software List. [On-line]. Available WWW: http://www.infoforce.qc.ca/spyware/. [11/04/2001].

Kabay, M. (1998). ICSA White Paper on Computer Crime Statistics. [On-line]. Available WWW: http://www.icsa.net/library/research/#info

Martin, D., Rajagopalan, S., Rubin, A. (1997). Blocking Java Applets at the Firewall. [On-line]. Available WWW:

Microsoft Corporation, (1996a). Internet Component Download. [On-line]. Available WWW: http://www.microsoft.com

Microsoft Corporation, (1996b). Microsoft Autehnticode Technology. [On-line]. Available WWW: http://www.microsoft.com

Microsoft Corporation, (1996c). Microsoft Internet Security Framework. [On-line]. Available WWW: http://www.microsoft.com

Microsoft Corporation, (1996d). OLE Controls 96. [On-line]. Available WWW: http://www.microsoft.com

Microsoft Corporation, (1996e). OLE Controls/COM Objects for the Internet - Draft 4. [On-line]. Available WWW: http://www.microsoft.com

Microsoft Corporation, (1996f). What Is the Exploder Control and How Does It Relate to Authenticode? [On-line]. Available WWW: http://www.microsoft.com

Miller, M.A (1994). Troubleshooting TCP/IP Analyzing the Protocols of the Internet. San Mateo: M&T Books.

Morar, J., Chess, D. (1998). Web Browsers – Threat or Menace? [On-line]. Available WWW: http://www.av.ibm.com/InsideTheLab/Bookshelf/ScientificPapers/Chess/Threate/Threat.html

Murhammer, M., Atakan, O., Bretz, L., Suzuki, K., Wood, D. (1998). TCP/IP Tutorial and Technical Overview. [On-line]. Available WWW: http://www.ibm.com

Nachenberg, C., Chien, E., Trilling, S. (1998). JavaApp.Strange Brew. [On-line]. Available WWW: http://www.symantec.com/avcenter/venc/data/javaapp.strangebrew.html.

Nachenberg, C. (1999). JavaApp.BeanHive. [On-line]. Available WWW: http://www.symantec.com/avcenter/venc/data/javaapp.beanhive.html.

Network Working Group (1999). Hypertext Transfer Protocol -- HTTP/1.1. [On-line]. Available WWW: http://www.w3c.org

NCompass Labs Inc, (1999). Authoring ActiveX Controls for the ScriptActive Plug-in. [On-line]. Available WWW: http://www.ncompasslabs.com/Plug-Ins/Documentation/Authoring+ ActiveX+Controls.htm

Oaks, S. (1998). Java Security. O'Reilly & Associates.

Office of the Federal Privacy Commissioner. (2000a). Privacy in Australia. [On-line]. Available WWW: http://www.privacy.gov.au/publications/pia.pdf. Downloaded: 8/12/2000.

Office of the Federal Privacy Commissioner. (2000b). Fact Sheet 2 – National Privacy Principles (Npps). [On-line]. Available WWW: http://www.privacy.gov.au/publications/fs2.pdf [8/12/2000.]

Office of the Federal Privacy Commissioner. (2000c). Fact Sheet 1 - Overview. [On-line]. Available WWW: http://www.privacy.gov.au/publications/fs1.pdf. [8/12/2000.]

Office of the Federal Privacy Commissioner. (2000d). Fact Sheet 3 - Codes. [On-line]. Available WWW: http://www.privacy.gov.au/publications/fs1.pdf. [8/12/2000].

Office of the Federal Privacy Commissioner. (2000e). Fact Sheet 4 - Powers. [On-line]. Available WWW: http://www.privacy.gov.au/publications/fs4.pdf. [8/12/2000].

Office of the Federal Privacy Commissioner. (2001f). Information Privacy Principles under the *Privacy Act 1988.* [On-line]. Available WWW http://www.privacy.gov.au/publications/ipps.html. [12/04/2001].

Office of the Federal Privacy Commissioner. (2001g). Privacy & the Public Sector. [On-line]. Available WWW http://www.privacy.gov.au/public/index.html. [12/04/2001].

Ogilvie, E. (2000). Cyberstalking. [On-line]. Available WWW http://www.privacy.gov.au/public/index.html. [11/0520/01].

Reynolds, J. (1989). RFC:1135 The Helminthiasis of the Internet. [On-line]. Available WWW: http://sunsite.hr/rfc/index_fr.html. [10/02/2000].

Somar Organisation. (1996). Windows NT Security Issues [On-line]. Available WWW: http://www.somar.com/security.html

Sun Micrososystems. (1999a). 100% Pure Java™ Certification Program. [On-line]. Available WWW: http://java.sun.com/100percent/ [04/02/2000].

Sun Micrososystems. Java Security Story. [On-line]. Available WWW: http://www.sun.com

Sun Micrososystems. The Java Language - An Overview. [On-line]. Available WWW: http://www.sun.com

Sun Micrososystems. (2000a). Chronology of security-related bugs and issues, 02/26/00. [On-line]. Available WWW: http://java.sun.com/sfaq/chronology.html

Sun Micrososystems. (2000b). Products & APIs. [On-line]. Available WWW: http://www.javasoft.com/products/

United States Justice Department. (1999). Cyberstalking: A New Challenge for Law Enforcement and Industry. [On-line]. Available http://www.usdoj.gov/criminal/cybercrime/cyberstalking.htm [11/05/2001].

Venners, B. (1997). Security and the Class Verifier. [On-line]. Available WWW: http://www.javaworld.com/javaworld/jw-10-1997/jw-10-hood.html. [25/05/1999]

World Wide Web Consortium. (1999). HTTP - Hypertext Transfer Protocol Overview. [On-line]. Available WWW: http://www.w3.org/Protocols/. [10/02/2000]

Yellin, F. (1996). Low Level Security in Java. [On-line]. Available WWW: http://www.sun.com