**Luanne walkthrough**

# **Index**

[Index 1](#_Toc169434426)

[List of pictures 1](#_Toc169434427)

[Disclaimer 2](#_Toc169434428)

[Reconnaissance 2](#_Toc169434429)

[Initial foothold 2](#_Toc169434430)

[User flag 5](#_Toc169434431)

[Privilege escalation 8](#_Toc169434432)

[APPENDIX A - CVE 11](#_Toc169434433)

[CVE-2010-2320 11](#_Toc169434434)

[APPENDIX B - GNUPG 11](#_Toc169434435)

[APPENDIX B - PGP 12](#_Toc169434436)

# **List of pictures**

[Figure 1 - nMap scan results 2](#_Toc169434443)

[Figure 2 - Credentials for web application on port 9001 found 3](#_Toc169434444)

[Figure 3 - Successful login on web application running on port 9001 3](#_Toc169434445)

[Figure 4 - Process logs 3](#_Toc169434446)

[Figure 5 - Robots.txt file on web application running on port 80 4](#_Toc169434447)

[Figure 6 - Content found on path /weather on web application running on port 80 4](#_Toc169434448)

[Figure 7 - City list 5](#_Toc169434449)

[Figure 8 - LUA command injection to obtain a shell 5](#_Toc169434450)

[Figure 9 - Shell obtained on the server 6](#_Toc169434451)

[Figure 10 - Hash found 6](#_Toc169434452)

[Figure 11 - Hash cracked 6](#_Toc169434453)

[Figure 12 - Process analysis 7](#_Toc169434454)

[Figure 13 - Local server information 7](#_Toc169434455)

[Figure 14 - RSA keys retrieved 8](#_Toc169434456)

[Figure 15 - RSA permissions 8](#_Toc169434457)

[Figure 16 - SSH connection 8](#_Toc169434458)

[Figure 17 - Interesting directories 9](#_Toc169434459)

[Figure 18 - Interesting file 9](#_Toc169434460)

[Figure 19 - PGP keys 9](#_Toc169434461)

[Figure 20 - .enc file decryption 9](#_Toc169434462)

[Figure 21 - New hashed credentials 10](#_Toc169434463)

[Figure 22 - Password cracked 10](#_Toc169434464)

[Figure 23 - Privilege escalation 10](#_Toc169434465)

[Figure 24 - Root flag 11](#_Toc169434466)

# **Disclaimer**

I do this box to learn things and challenge myself. I’m not a kind of penetration tester guru who always knows where to look for the right answer. Use it as a guide or support. Remember that it is always better to try it by yourself. All data and information provided on my walkthrough are for informational and educational purpose only. The tutorial and demo provided here is only for those who’re willing and curious to know and learn about Ethical Hacking, Security and Penetration Testing.

# **Reconnaissance**

The results of an initial nMap scan are the following:

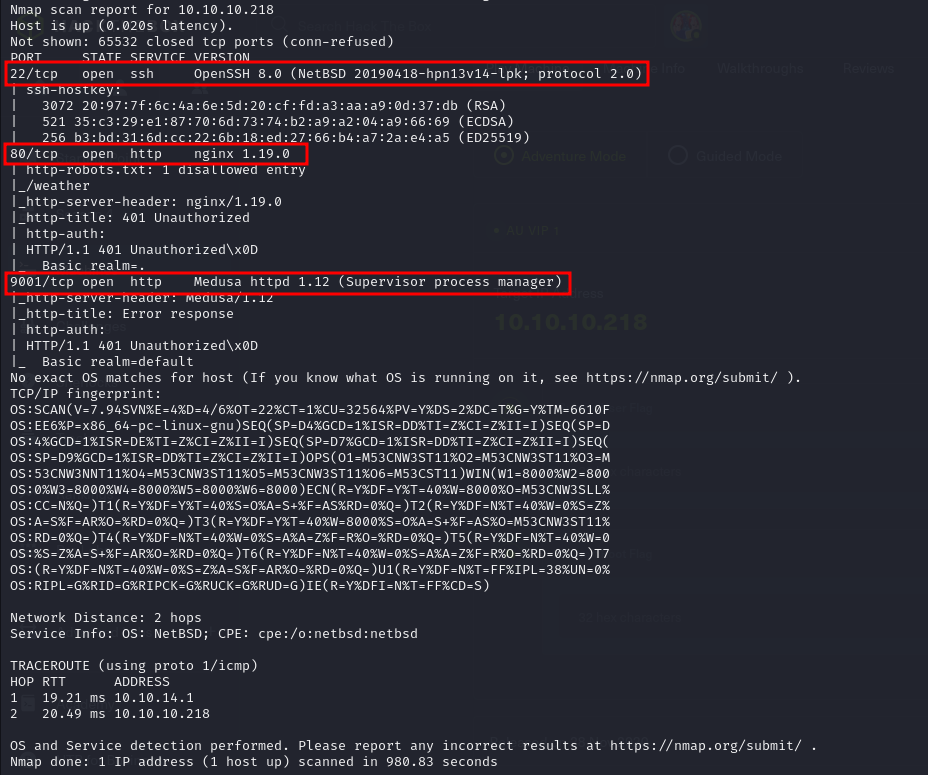


Figure 1 - nMap scan results

Open ports are 22, 80 and 9001. So, this box has SSH enabled, a web application running on port 80 and a different one running on port 9001. Also, nMap guessed the OS as NetBSD.

# **Initial foothold**

If I try to access to the web application on port 9001, it asks me credential. So, I tried to execute a brute force attack to find them running the following command:

Luckly, hydra found valid credentials!

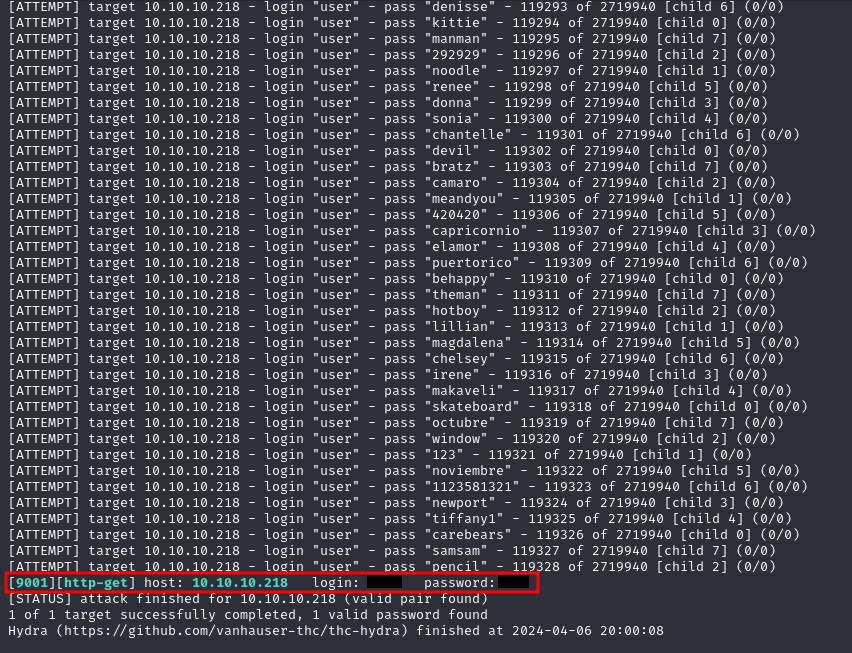


Figure 2 - Credentials for web application on port 9001 found

The private area I can access shows me information about the server as memory used, processes and uptime, as shown in the following picture:

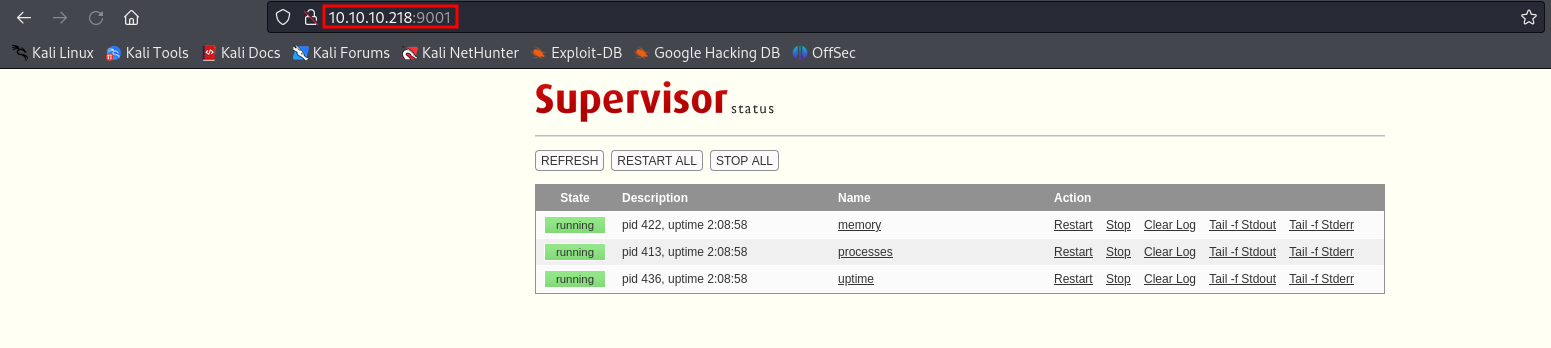


Figure 3 - Successful login on web application running on port 9001

Also, I understand that the server run LUA scripts analyzing the process logs:

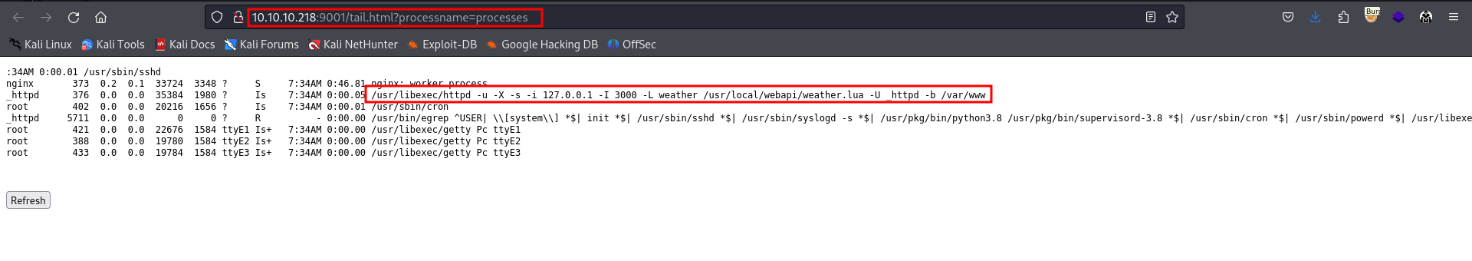


Figure 4 - Process logs

On the other hand, analyzing the web application on port 80, I found that robots.txt file contains an interesting path to explore:

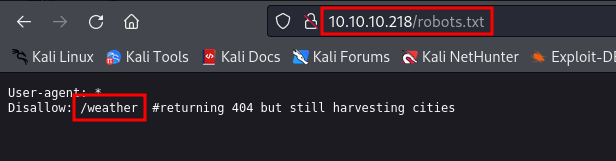


Figure 5 - Robots.txt file on web application running on port 80

However this path is partial and it is not usable in this way. So, I run DirBuster to find content in this path:

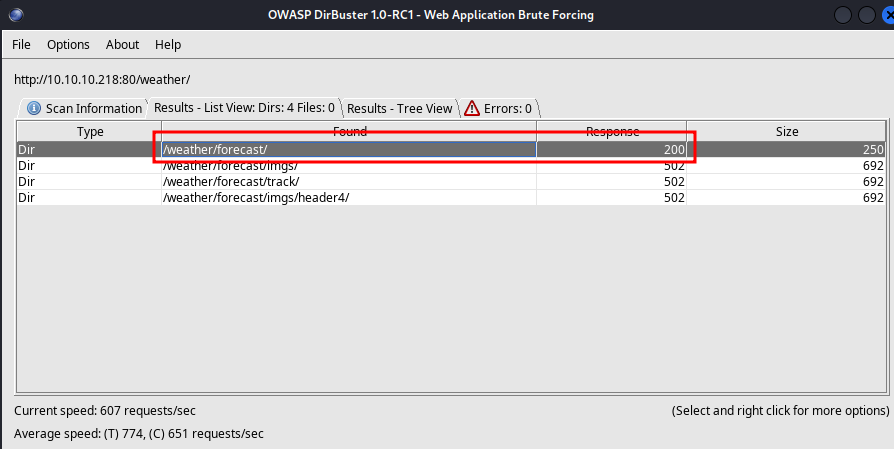


Figure 6 - Content found on path /weather on web application running on port 80

The new path found uses API to retrieve weather information about a city passed as parameter. A list of cities accepted is:

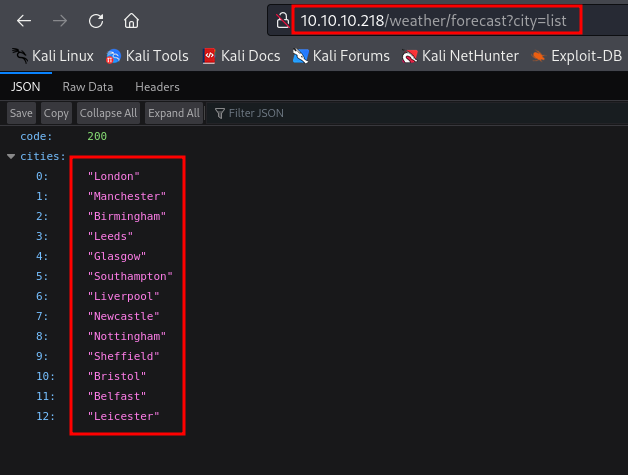


Figure 7 - City list

# **User flag**

Exploiting the API found, I can inject code. In particular, I need to inject LUA code (in fact I found that server run LUA scripts). So, I intercept a request to this API and I inject my payload using Burp Suite:



Figure 8 - LUA command injection to obtain a shell

Of course, I need to open a listener on the port specified in the command and I can obtain a shell:

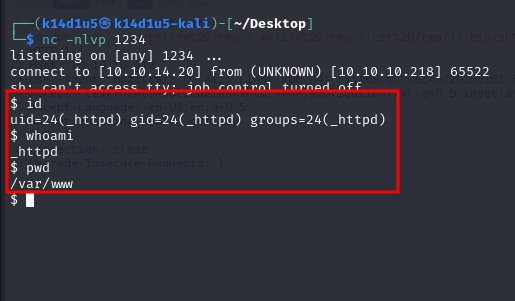


Figure 9 - Shell obtained on the server

However, I obtain a shell with user **\_httpd**. This means that I need to perform a lateral movement to became a different user and retrieve the user flag. In the folder I have the shell, **/var/www**, I found a file named **.htpasswd**. Analyzing this file, I found a hashed credential:

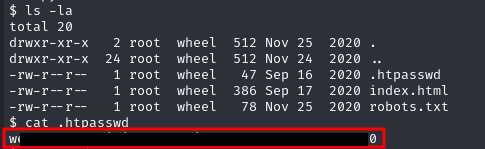


Figure 10 - Hash found

So, I tried to crack this hash running the following command:

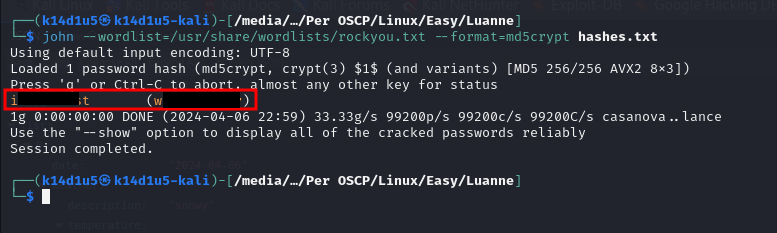


Figure 11 - Hash cracked

Luckly, I cracked the credentials! The first thing I tried was using these to access to SSH, but them didn’t work. SSH ask an RSA key to login. So, I tried to use them to login on the web application running on port 80. This time them worked! However, at this point I was lost. So, I came back on my shell to search new information. If I run Linpeas, or if I check myself the process as shown in the next figure, I found the following command run:

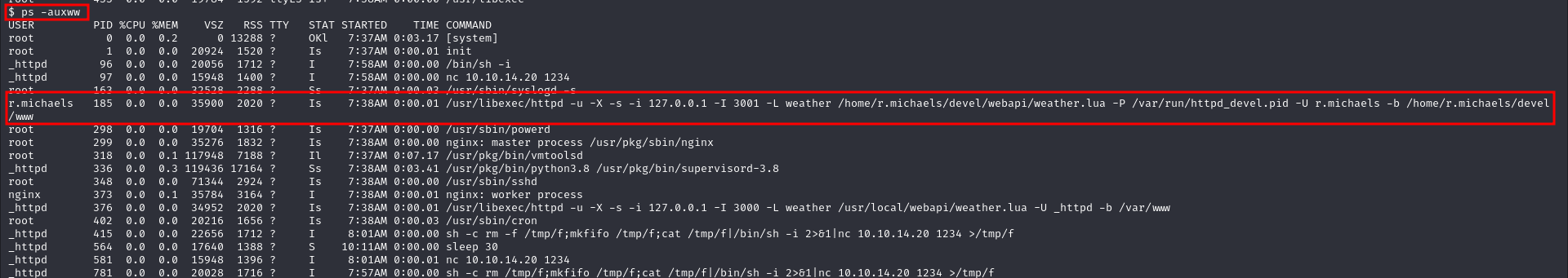


Figure 12 - Process analysis

I noted that this command is the same used by the web application and I found in the process logs from the browser. At this point, I interact with this service as the web application does and I analyze the interaction using the **curl** command:

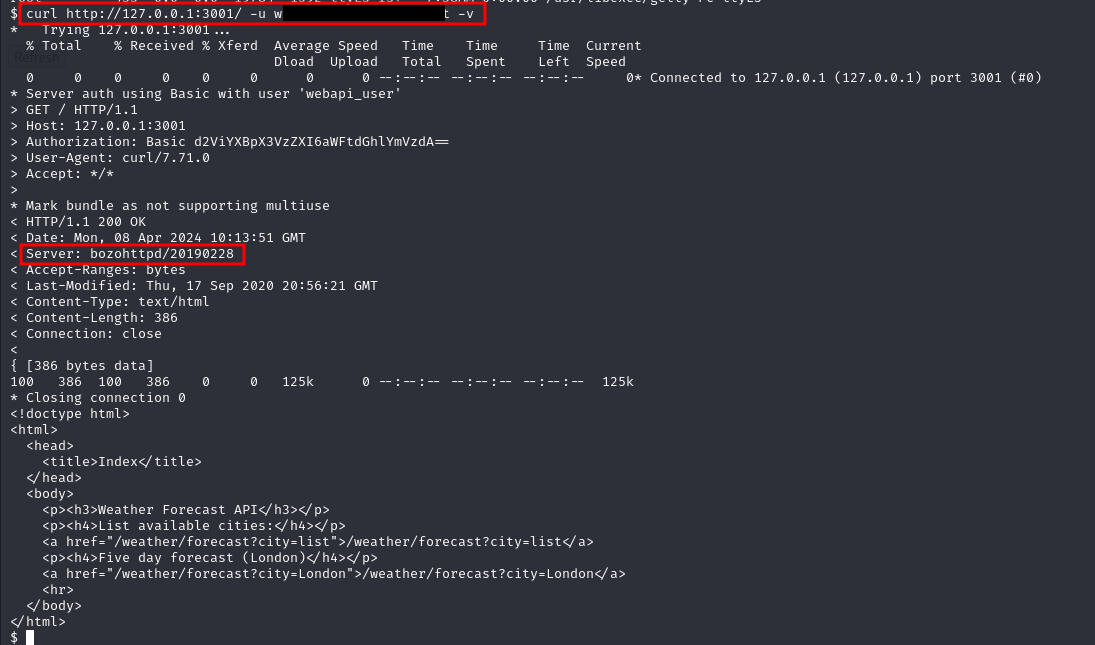


Figure 13 - Local server information

Obviously, I need to provide the credentials I found previously to access to the application. Thanks to this interaction, I found what is the local server. Looking for it on Google, I found out that it is vulnerable to **CVE-2010-2320**. It allows to retrieve the RSA keys from the home directory of the user (**r.michaels** in this case). To do it, I run the command in the following figure:

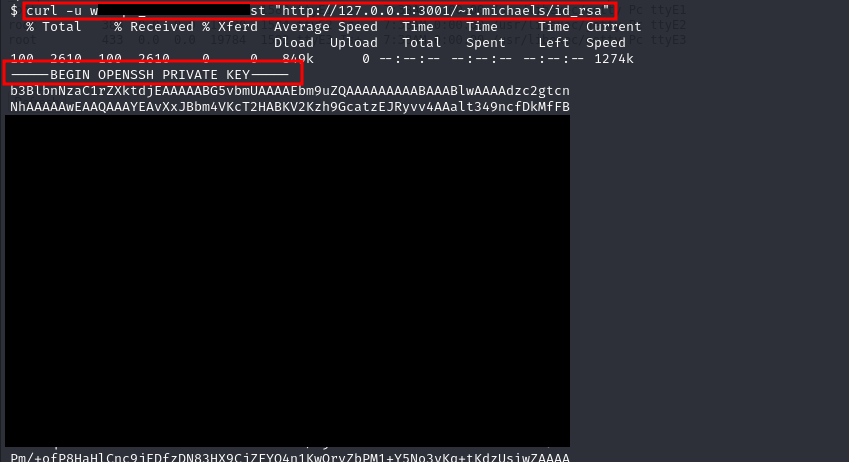


Figure 14 - RSA keys retrieved

I copied it in a file and, before can use it, I need to set the right permission on it:



Figure 15 - RSA permissions

Finally, I used it to log in as **r.michaels** and retrieve the user flag (I forgot the user flag screen)!

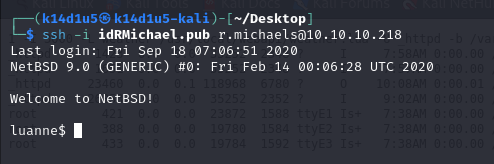


Figure 16 - SSH connection

# **Privilege escalation**

At this point I finally can think how to escalate my privilege to root user. Analyzing the **r.michaels** home directory, I found two interesting directories:

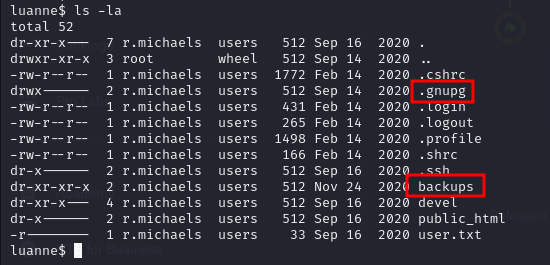


Figure 17 - Interesting directories

In particular, I found an interesting **.enc** file in the **backups** folder.

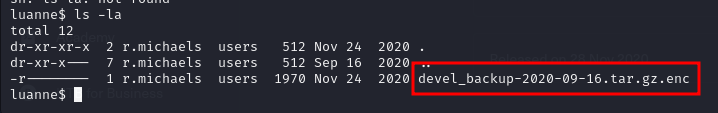


Figure 18 - Interesting file

This file can be decrypted using a PGP key, usually stored in the **.gnupg** in the user home directory:

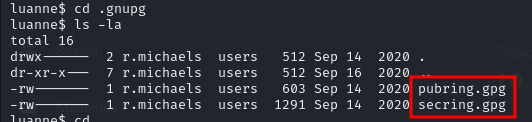


Figure 19 - PGP keys

So, on the target machine, I can run the following command:

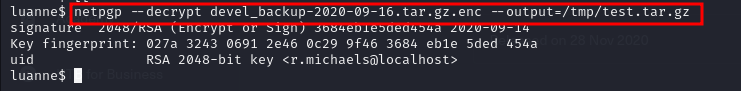


Figure 20 - .enc file decryption

Actually, I can’t use the **/tmp/** directory to store the output file because them will be automatically deleted after few seconds. So, I give permission on the user home directory and I actually used it instead. Similar to what I did previously, I found a new **.htpasswd** in the **www** directory of the backup decrypted files. Again, this file contains a hashed credential:

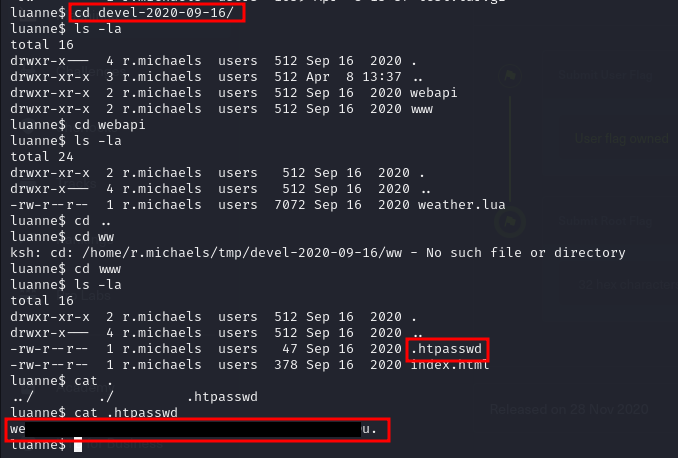


Figure 21 - New hashed credentials

As before, I try to crack them running the following command:

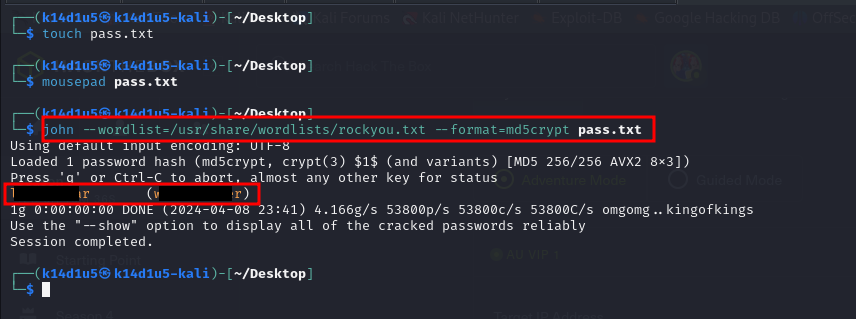


Figure 22 - Password cracked

To became root user, I remembered that nMap told me the OS was NetBSD, so I used the appropriate command became root:

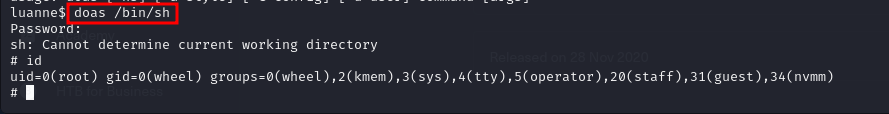


Figure 23 - Privilege escalation

Now, I just have to retrieve the root flag:



Figure 24 - Root flag

# **APPENDIX A - CVE**

## **CVE-2010-2320**

bozotic HTTP server (aka bozohttpd) before 20100621 allows remote attackers to list the contents of home directories, and determine the existence of user accounts, via multiple requests for URIs beginning with /~ sequences. The server is not properly handling requests to a user’s public\_html folder while the folder does not exist. This can be exploited to determine the existence of user accounts via multiple requests for URIs beginning with /~ sequences. Successful exploitation will allow attacker to determine the existence of a user and potentially disclose the user’s files.

Bozohttpd is started from inetd with a configuration line in /etc/inetd.conf like this:

There is a ~user1/public\_html and there are other users on the system but without a public\_html.

1) Go to "http://localhost/~user1/"

I get the index.html from user1/public\_html as expected

2) Go to "http://localhost/~user2/" (who don't have a public\_html dir)

I get a "403 Forbidden /~user2/: Access to this item has been denied", as expected

3) Go to "http://localhost/~user2/" again (reload the page)

I don't get the error above, but just the directory index of ~user2 (/home/user2).

If I reload the page, I get the result of 2) and 3) swapping around. 3) Shouldn't happen, as there is no public\_html there. And anyone can:

a) Probe for user names in the system (dir is there or not)

b) Look at least the name of the files of some user.

# **APPENDIX B - GNUPG**

GnuPG is a complete and free implementation of the OpenPGP standard as defined by RFC4880 (also known as PGP). GnuPG allows you to encrypt and sign your data and communications; it features a versatile key management system, along with access modules for all kinds of public key directories. GnuPG, also known as GPG, is a command line tool with features for easy integration with other applications. A wealth of frontend applications and libraries are available. GnuPG also provides support for S/MIME and Secure Shell (ssh).

Since its introduction in 1997, GnuPG is Free Software (meaning that it respects your freedom). It can be freely used, modified and distributed under the terms of the GNU General Public License .

Gpg4win is a Windows version of GnuPG featuring a context menu tool, a crypto manager, and an Outlook plugin to send and receive standard PGP/MIME mails.

# **APPENDIX B - PGP**

Pretty Good Privacy (PGP) is an encryption system used for both sending encrypted emails and encrypting sensitive files. Since its invention back in 1991, PGP has become the de facto standard for email security.

The popularity of PGP is based on two factors. The first is that the system was originally available as freeware, and so spread rapidly among users who wanted an extra level of security for their email messages. The second is that since PGP uses both symmetric encryption and public-key encryption, it allows users who have never met to send encrypted messages to each other without exchanging private encryption keys.

There are, essentially, three main uses of PGP:

* Sending and receiving encrypted emails.
* Verifying the identity of the person who has sent you this message.
* Encrypting files stored on your devices or in the cloud.

Of these three uses, the first – sending secure email – is by far the dominant application of PGP.

The major pro of PGP encryption is that it is essentially unbreakable. That’s why it is still used by journalists and activists, and why it is often regarded as the best way of improving cloud security. In short, it is essentially impossible for anyone – be they a hacker or even the NSA – to break PGP encryption.

Though there have been some news stories that point out security flaws in some implementations of PGP, such as the Efail vulnerability, it’s important to recognize that PGP itself is still very secure.

The biggest con of PGP encryption is that it is not that user-friendly. This is changing – thanks to off-the-shelf solutions that we will come to shortly – but using PGP can add significant extra work and time to your daily schedule. In addition, those using the system need to be aware of how it works, in case they introduce security holes by using it incorrectly. This means that businesses considering a move to PGP will need to provide training.

For that reason, many businesses might want to consider alternatives. There are encrypted messaging apps like Signal, for instance, that offer encryption that is more straightforward to use. In terms of storing data, anonymisation can be a good alternative to encryption and can be a more efficient use of resources.

Finally, you should be aware that PGP encrypts your messages, but it doesn’t make you anonymous. Unlike anonymous browsers using proxy servers or working through a VPN to hide your true location, emails sent through PGP can be traced to a sender and recipient. Their subject lines are not encrypted either, so you shouldn’t put any sensitive information there.