Hack The Box Cap Walkthrough

Note: This is an easy box (but considerably hard for a complete newbie). Always count from zero like a real Captain!!!! (This will unfold later)

Step1: Enumeration

A fast NMAP Scan gives us three ports open as follows:

21→vstfpd 3.0.3

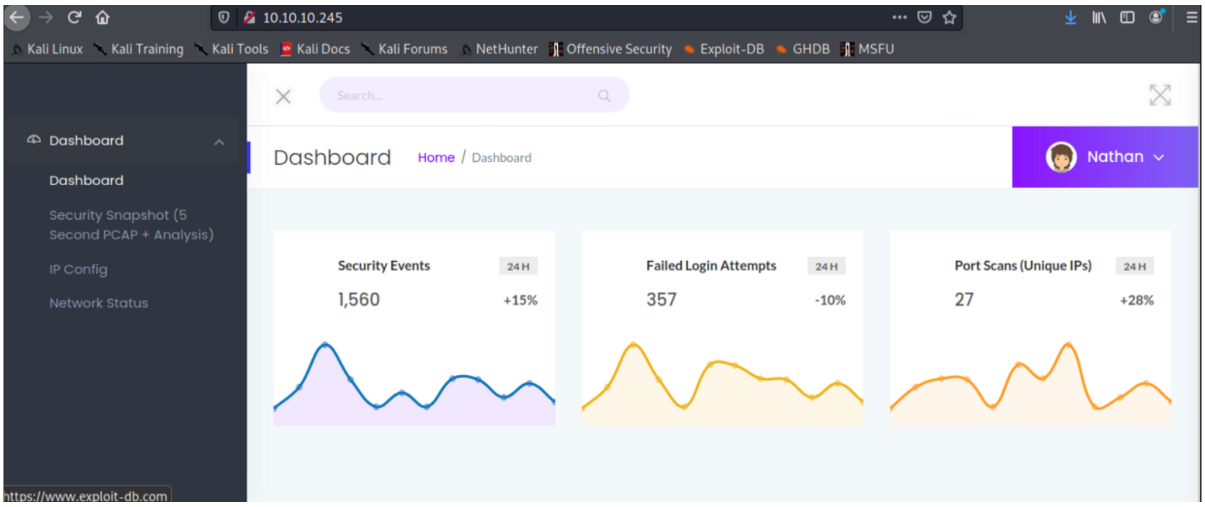
22→OpenSSH 8.2p1

80→http

Out of the above three, the first two are not significant for pwning as we do not have the credentials and anonymous login is also not allowed. But as we can see there is a http service running on port 80.

Step2: Enumerating service on port 80

We need to perform more recon on this so we navigate to our browser and on the URL http://10.10.10.245:80 we get a dashboard that displays some statistics. Upon clicking some clickable links we stumble upon the Download button that redirects us to a very specific URL and downloads a file named “10.pcap” for us.(The numbers game starts from here 😊) This file is completely empty which should trick our mind to count from zero.



Step3: Counting from zero 😊

The name of the machine is Captain and so we now unfold the secret. Here as seen before that a file named “10.pcap” is downloaded and now instead of being redirected to “http:10.10.10.10.245/data/10” we will manually go to “http:10.10.10.10.245/data/0” and this will download a file “0.pcap” for us.

Step4: Inspecting the new pcap file

This pcap file unlike the “10.pcap” file has packets captured in it. These packets are transferred between the user “nathan” when he is communicating/logging into the dashboard that we see.

Upon digging deep in the new file using wireshark line by line we can see that there are specific plain text fields spilling out username and password. Come on Be Quick NOTE IT DOWN!!!!

Step5: Putting the creds to use

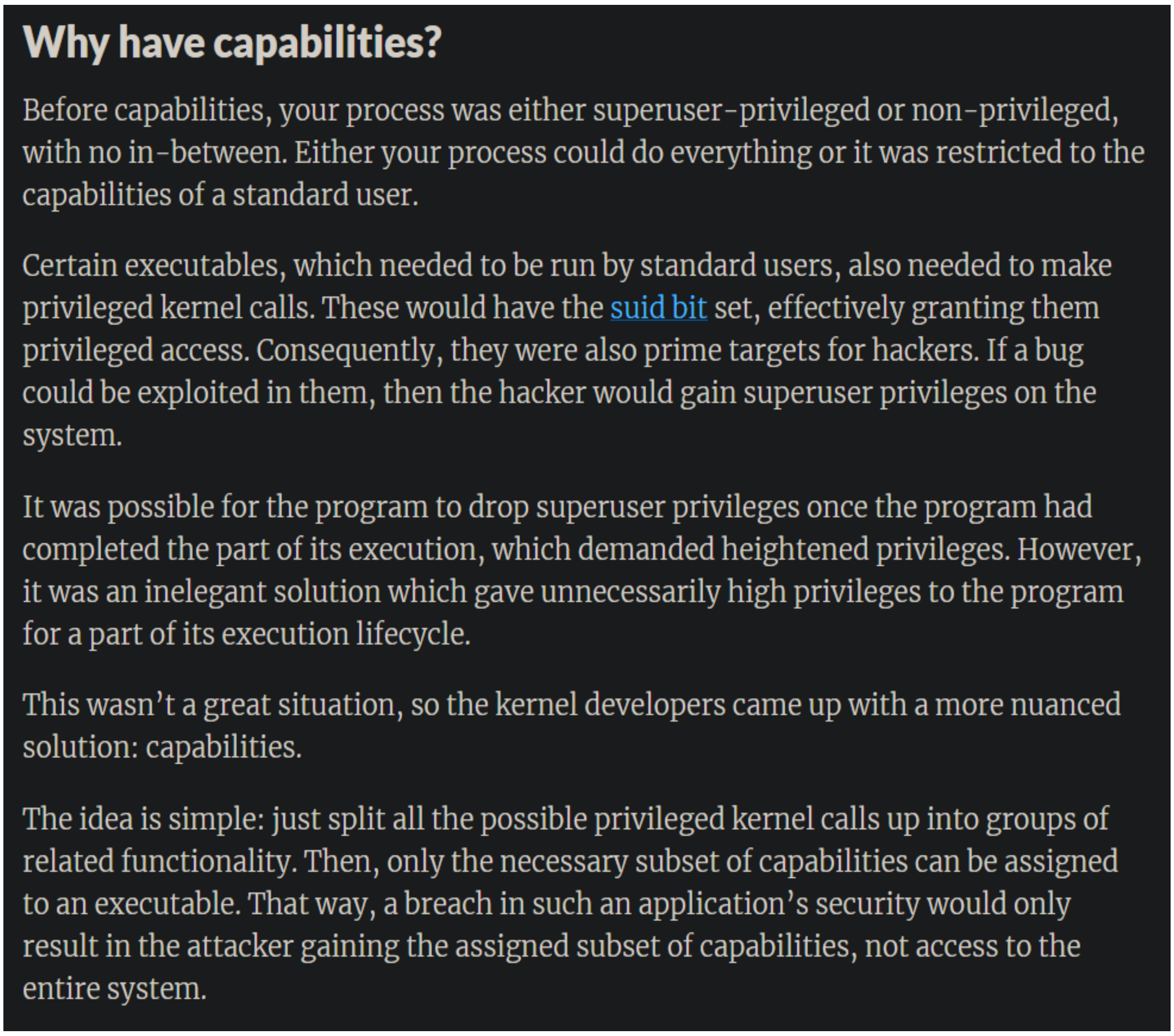
We have found complete information necessary to login 'somewhere' as the low level user "nathan", but where do we use it??!! If you remember we found a SSH service running on the machine. Try it there and you will be able to login as the low-level user.

Note: You can also use the SSH User Enumeration method to confirm whether the user Nathan actually exists on the machine and whether we can login using ssh. (We can also login using port 21 as we now have the creds)

Step6: Enumerating the local user files and PE

Upon changing into various directories, we get the user flag. Upon checking which directories are accessible for the low-level user we see that we are not allowed to change into the "root" directory. This tells us that there is no simple way to get root as the permissions are properly set as of now.

When we try changing the permission for root we get "Permission denied" error. Linux capabilities is the answer for us to get root.



Why have capabilities?

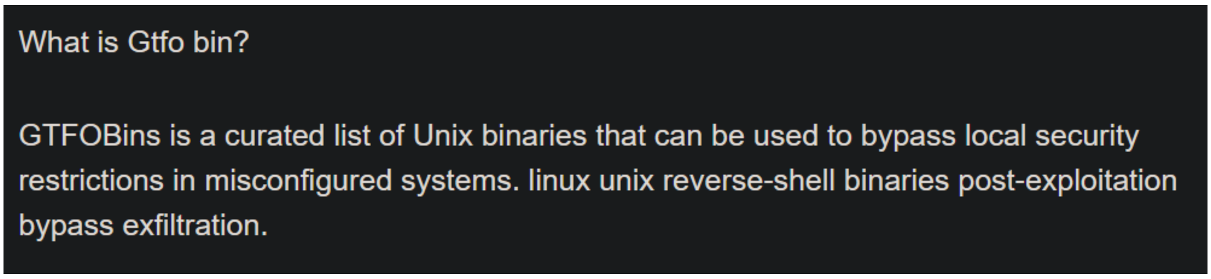
Before capabilities, your process was either superuser-privileged or non-privileged, with no in-between. Either your process could do everything or it was restricted to the capabilities of a standard user.

Certain executables, which needed to be run by standard users, also needed to make privileged kernel calls. These would have the suid bit set, effectively granting them privileged access. Consequently, they were also prime targets for hackers. If a bug could be exploited in them, then the hacker would gain superuser privileges on the system.

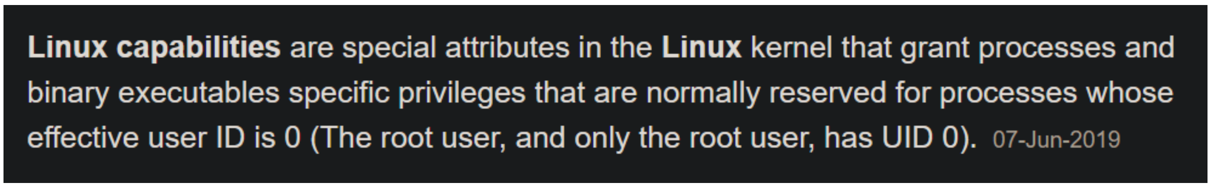
It was possible for the program to drop superuser privileges once the program had completed the part of its execution, which demanded heightened privileges. However, it was an inelegant solution which gave unnecessarily high privileges to the program for a part of its execution lifecycle.

This wasn't a great situation, so the kernel developers came up with a more nuanced solution: capabilities.

The idea is simple: just split all the possible privileged kernel calls up into groups of related functionality. Then, only the necessary subset of capabilities can be assigned to an executable. That way, a breach in such an application's security would only result in the attacker gaining the assigned subset of capabilities, not access to the entire system.

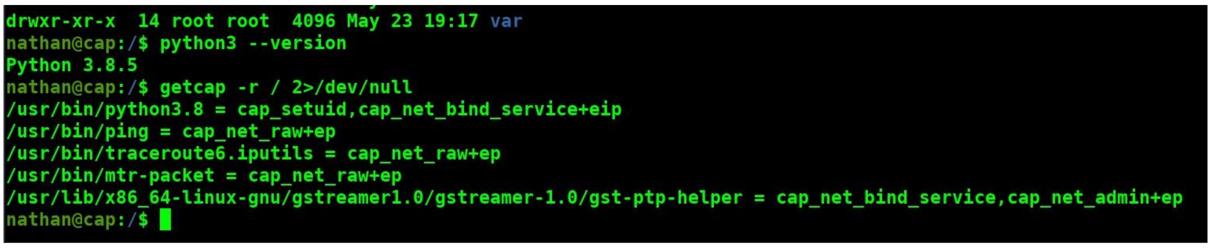

What is Gtfo bin?

GTFOBins is a curated list of Unix binaries that can be used to bypass local security restrictions in misconfigured systems. linux unix reverse-shell binaries post-exploitation bypass exfiltration.


Linux capabilities are special attributes in the **Linux** kernel that grant processes and binary executables specific privileges that are normally reserved for processes whose effective user ID is 0 (The root user, and only the root user, has UID 0). 07-Jun-2019

Useful Link:

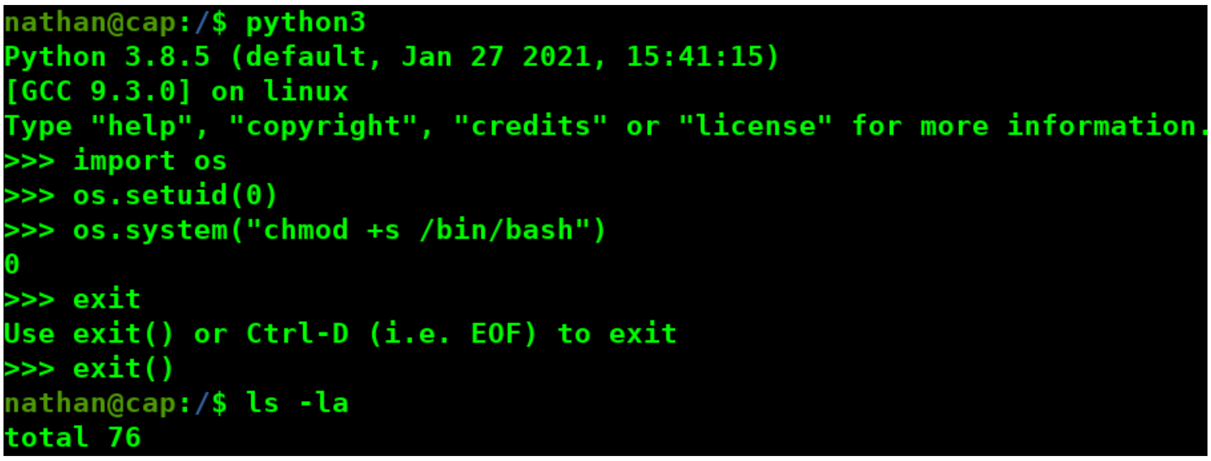
https://gtfobins.github.io/


```
drwxr-xr-x  14 root root  4096 May 23 19:17 var
nathan@cap:/$ python3 --version
Python 3.8.5
nathan@cap:/$ getcap -r / 2>/dev/null
/usr/bin/python3.8 = cap_setuid,cap_net_bind_service+eip
/usr/bin/ping = cap_net_raw+ep
/usr/bin/traceroute6.iputils = cap_net_raw+ep
/usr/bin/mtr-packet = cap_net_raw+ep
/usr/lib/x86_64-linux-gnu/gstreamer1.0/gstreamer-1.0/gst-ptp-helper = cap_net_bind_service,cap_net_admin+ep
nathan@cap:/$ 
```

Getcap gives us the capabilities for each file in any directory that we specify, here *root.* Here the stderr (standard error) will produce a lot of *Failed to get capabilities of file* errors so we are redirecting the errors to */dev/null*. 2 is the notation for stderr and so it is *2>/dev/null.* The errors get directed to the null file and only the results having the capabilities enabled are shown.

Here the python3.8 executable has the capabilities and we use the same to do PrivEsc. Since python has the permission to run commands with sudo rights even though the commands are coming from a low-level user, we are going to use python commands and first set the uid to 0, then set the gid for */bin/bash* so that we can run it as root.

We can do it using the "/bin/bash -p" command but we currently do not have permissions to do so. We will use python do PE. We first open a python interpreter then proceed as follows:


```
nathan@cap:/$ python3
Python 3.8.5 (default, Jan 27 2021, 15:41:15)
[GCC 9.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import os
>>> os.setuid(0)
>>> os.system("chmod +s /bin/bash")
0
>>> exit
Use exit() or Ctrl-D (i.e. EOF) to exit
>>> exit()
nathan@cap:/$ ls -la
total 76
```

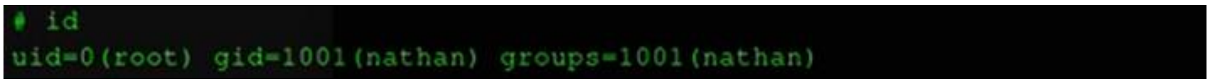
There is a lot going on here, let us understand it bit by bit.

1. Using the interpreter, we import the os module

(The os module is Python's answer to handling many Linux file operations.The os module allows Python to perform many of the file and folder operations that you'd typically carry out in the Linux command line. It enable you to begin swapping out Bash for Python, which makes for a much cleaner and friendlier scripting experience. More about it at: https://linuxconfig.org/python-os-module)

2. We set the uid to "0" meaning we set the user_id to that of root_user which is always "0"
3. The third command will type the "chmod +s /bin/bash" command onto the terminal ie. It is the same as typing the command in the terminal.
4. The command that is used will set the g_id (group_id) and will give us execute permissions to /bin/bash shell.
5. We then exit the python interpreter using the exit function

Result of the above steps:



```
# id
uid=0(root) gid=1001(nathan) groups=1001(nathan)
```

Note: https://www.liquidweb.com/kb/how-do-i-set-up-setuid-setgid-and-sticky-bits-on-linux/ (read more about file permissions and setuid, setgid here)

Then we spawn a /bin/bash shell onto the vulnerable machine using the "/bin/bash -p" command. Here the -p switch instructs the system to open the bash shell in a POSIX format. POSIX is a standard way just like we have OSI model for communication.

Note: More about POSIX at: https://linuxconfig.org/python-os-module

Now in the newly spawned POSIX shell we are able to view and also list the contents of the root folder and there we get our root flag.

ENJOY!!!!!