PROJECT 2 REPORT

CS4371

Instructor: Dr. Randy Klepetko

Mar 28, 2023

PROJECT 2

BY

Lauren Taylor

Dillon Hughes

Stanley Nwajiaku

Section I (Introduction):

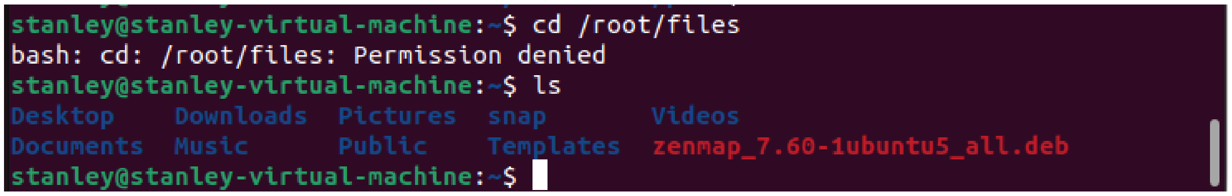
In this project, we have scenarios and tasks related to computer security and vulnerability exploitation. The tasks involve exploring and testing vulnerabilities in two different systems, A.1 and A.2, and exploiting these vulnerabilities to gain access to sensitive information and execute arbitrary code. Additionally, we examine various defense mechanisms, such as randomization of the stack memory address space, and discuss their effectiveness in preventing exploitation. As a team, we worked together to complete these tasks and explore the different vulnerabilities and defense mechanisms, highlighting the importance of secure computer systems and the measures necessary to protect against malicious attacks.

The task involves preparing target programs, analyzing the echo program for vulnerabilities, and exploiting the target programs. Firstly, a non-root user is created on the Ubuntu server and Kali machine, and the project files are uploaded and compiled using the provided Makefile. The echo program is started on the Ubuntu server and checks if it is running smoothly by connecting to it from the Kali machine. Then, the Damn Vulnerable Web Application is installed on the Ubuntu machine, and the web service is accessed from another machine to login as an admin. In the second task, the tcph program is built on the Kali and Ubuntu machines, and gdb is used to analyze tcph by setting a breakpoint in foo() and finding $rsp, $rbp, buf, and the return address of foo(). Finally, in the third task, the echo program is exploited by editing the attack.c program and building the exploit program to capture the hacking packet using Wireshark to find the files in /root/files in A.1. Additionally, the DVWA website is exploited by setting the script security to medium, reading the source code of the SQL injection, and injecting SQL statements to obtain all user IDs, first names, and last names using union and select SQL statements.

Section II (Task II):

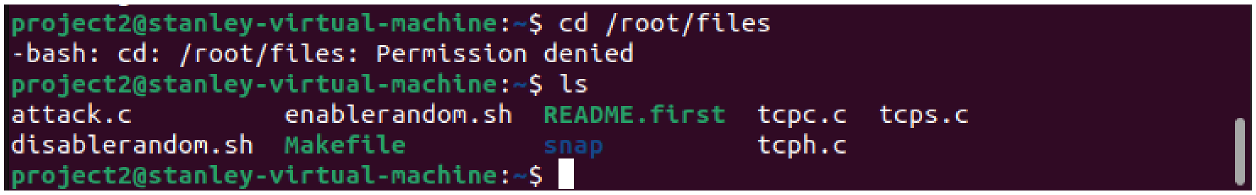
(i) Show whether or not you can read the files in /root/files of A.1 with local login and SSH login.

<u>For local login</u>



```
stanley@stanley-virtual-machine:~$ cd /root/files
bash: cd: /root/files: Permission denied
stanley@stanley-virtual-machine:~$ ls
Desktop    Downloads  Pictures  snap       Videos
Documents  Music      Public    Templates  zenmap_7.60-1ubuntu5_all.deb
stanley@stanley-virtual-machine:~$
```

<u>For SSH login</u>

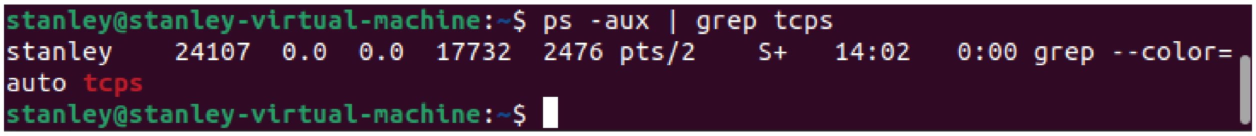


```
project2@stanley-virtual-machine:~$ cd /root/files
-bash: cd: /root/files: Permission denied
project2@stanley-virtual-machine:~$ ls
attack.c          enablerandom.sh  README.first  tcpc.c  tcps.c
disablerandom.sh  Makefile         snap          tcph.c
project2@stanley-virtual-machine:~$
```

(ii) Find and report exactly how many bytes are needed to crash the echo program.

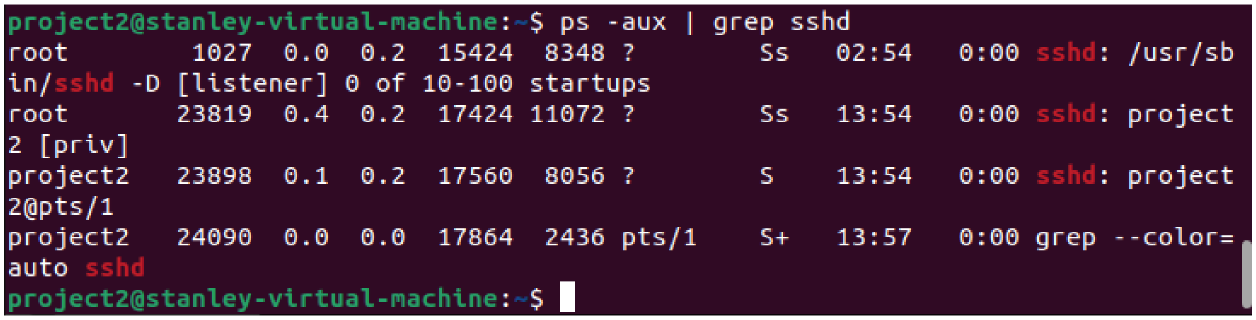
\x10 or 16 bytes past the location of buf.

(iii) Show which user ID is running the echo program in A.1.



```
stanley@stanley-virtual-machine:~$ ps -aux | grep tcps
stanley    24107  0.0  0.0  17732  2476 pts/2    S+   14:02   0:00 grep --color=
auto tcps
stanley@stanley-virtual-machine:~$
```

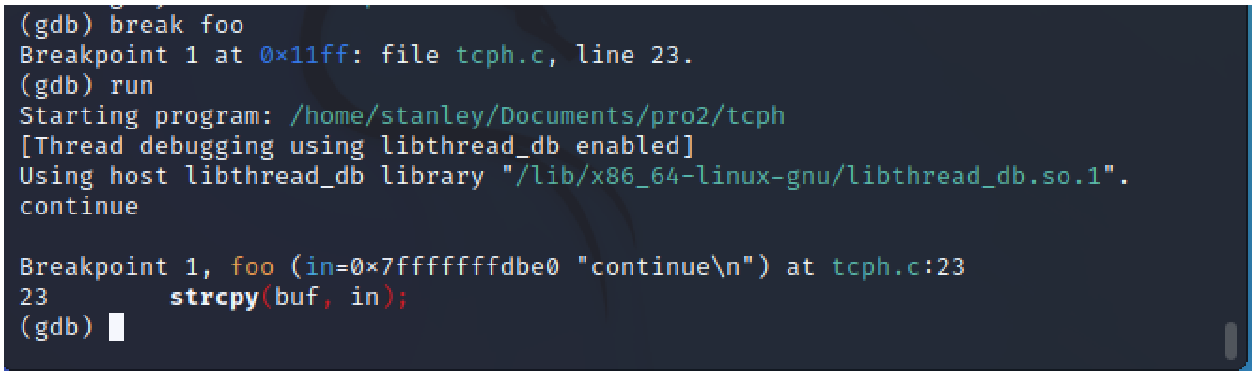
(iv) Show which user ID is running the SSH service in A.1.



```
project2@stanley-virtual-machine:~$ ps -aux | grep sshd
root        1027  0.0  0.2  15424  8348 ?        Ss   02:54   0:00 sshd: /usr/sb
in/sshd -D [listener] 0 of 10-100 startups
root       23819  0.4  0.2  17424 11072 ?        Ss   13:54   0:00 sshd: project
2 [priv]
project2   23898  0.1  0.2  17560  8056 ?        S    13:54   0:00 sshd: project
2@pts/1
project2   24090  0.0  0.0  17864  2436 pts/1    S+   13:57   0:00 grep --color=
auto sshd
project2@stanley-virtual-machine:~$
```

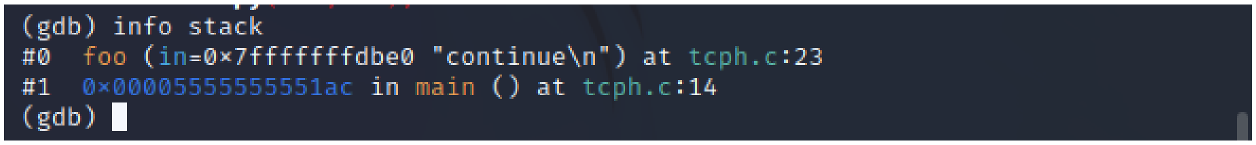
Section III (Task III):

(i) Show a screenshot of gdb running to a breakpoint in foo() of tcph in A.2.



```
(gdb) break foo
Breakpoint 1 at 0x11ff: file tcph.c, line 23.
(gdb) run
Starting program: /home/stanley/Documents/pro2/tcph
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".
continue

Breakpoint 1, foo (in=0x7fffffffdbe0 "continue\n") at tcph.c:23
23          strcpy(buf, in);
(gdb)
```

(ii) Show a screenshot of gdb showing the stack of foo() of tcph in A.2.



```
(gdb) info stack
#0  foo (in=0x7fffffffdbe0 "continue\n") at tcph.c:23
#1  0x00005555555551ac in main () at tcph.c:14
(gdb) 
```

(iii) Report the values of $rsp, $rbp, the address of buf, and the address of the return address of foo() in A.2.

- ❖ x/s $rsp: 0x7fffffffdbb0: ""
- ❖ x/s $rbp: 0x7fffffffdbd0: "\360\335\377\377\377\177"
- ❖ x/x $rbp+8: 0x7fffffffdbd8: 0xac

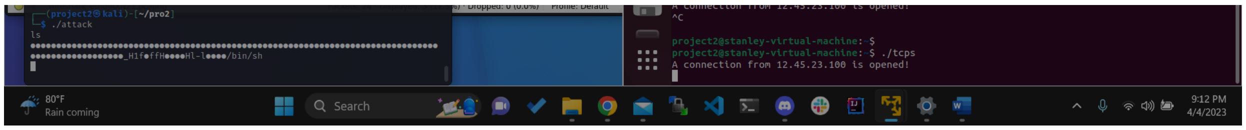
(iv) Report the values of $rsp, $rbp, the address of buf, and the address of the return address of foo() in A.1.

➔ After putting breakpoint in A.1 ubuntu terminal at tcph.c:23

- ❖ x/s $rsp: 0x7fffffffe190
- ❖ x/s $rbp: 0x7fffffffe1b0
- ❖ x/x $rbp+8: 0x7fffffffe1a8

Section IV (Task IV):

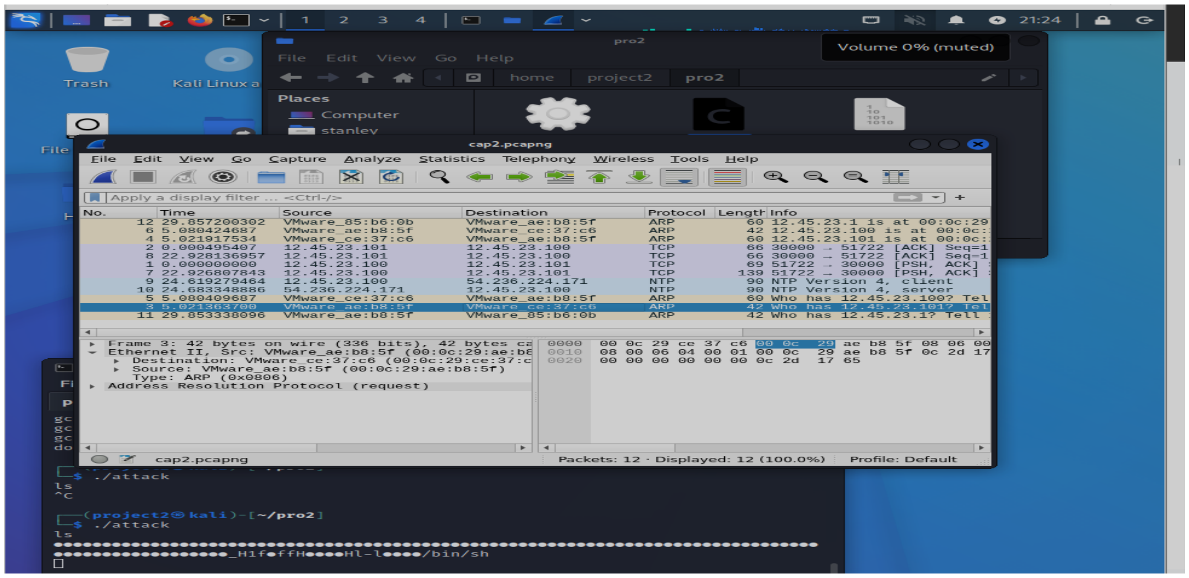
(i) Show a screenshot that the echo program is exploited.



```
┌──(project2㉿kali)-[~/pro2]
└─$ ./attack
ls
●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●
●●●●●●●●●●●●●●●●●_H1f●ffH●●●●Hl-l●●●●/bin/sh
```

```
A connection from 12.45.23.100 is opened!
^C

project2@stanley-virtual-machine:~$
project2@stanley-virtual-machine:~$ ./tcps
A connection from 12.45.23.100 is opened!
```

(ii) Show the exploiting packet captured in A.2.



```
No.  Time          Source              Destination         Protocol Length Info
 12 29.857200302  VMware_85:b6:0b     VMware_ae:b8:5f     ARP       60 12.45.23.1 is at 00:0c:29
  6 5.080424687   VMware_ae:b8:5f     VMware_ce:37:c6     ARP       42 12.45.23.100 is at 00:0c:
  4 5.021917534   VMware_ce:37:c6     VMware_ae:b8:5f     ARP       60 12.45.23.101 is at 00:0c:
  2 0.000495407   12.45.23.101        12.45.23.100        TCP       66 30000 → 51722 [ACK] Seq=1
  8 22.928136957  12.45.23.101        12.45.23.100        TCP       66 30000 → 51722 [ACK] Seq=1
  1 0.000000000   12.45.23.100        12.45.23.101        TCP       69 51722 → 30000 [PSH, ACK]
  7 22.926807843  12.45.23.100        12.45.23.101        TCP      139 51722 → 30000 [PSH, ACK]
  9 24.619279464  12.45.23.100        54.236.224.171      NTP       90 NTP Version 4, client
 10 24.683348886  54.236.224.171      12.45.23.100        NTP       90 NTP Version 4, server
  5 5.080409687   VMware_ce:37:c6     VMware_ae:b8:5f     ARP       60 Who has 12.45.23.100? Tel
  3 5.021363700   VMware_ae:b8:5f     VMware_ce:37:c6     ARP       42 Who has 12.45.23.101? Tel
 11 29.853338096  VMware_ae:b8:5f     VMware_85:b6:0b     ARP       42 Who has 12.45.23.1? Tell

▸ Frame 3: 42 bytes on wire (336 bits), 42 bytes ca
▾ Ethernet II, Src: VMware_ae:b8:5f (00:0c:29:ae:b8
   ▸ Destination: VMware_ce:37:c6 (00:0c:29:ce:37:c
   ▸ Source: VMware_ae:b8:5f (00:0c:29:ae:b8:5f)
     Type: ARP (0x0806)
▸ Address Resolution Protocol (request)

0000  00 0c 29 ce 37 c6 00 0c 29 ae b8 5f 08 06 00
0010  08 00 06 04 00 01 00 0c 29 ae b8 5f 0c 2d 17
0020  00 00 00 00 00 00 0c 2d 17 65

cap2.pcapng        Packets: 12 · Displayed: 12 (100.0%)        Profile: Default
```

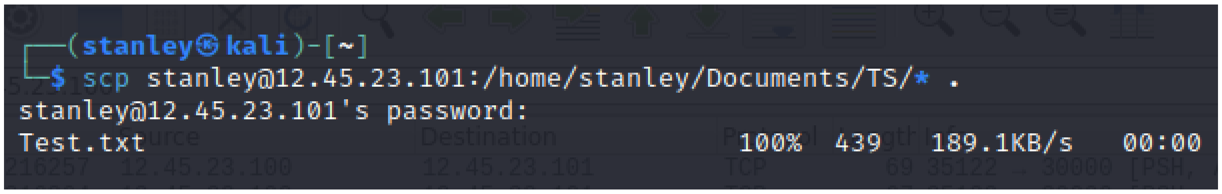
```
└─$ ./attack
ls
^C

┌──(project2㉿kali)-[~/pro2]
└─$ ./attack
ls
●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●
●●●●●●●●●●●●●●●●●●●_H1f●ffH●●●●Hl-l●●●●/bin/sh
```

(iii) Report how you retrieve the files from A.1 to A.2. Give steps in details.

- ❖ So to transfer a file from A.1 ubuntu vmware with ip address (12.45.23.101) into the A.2 Kali Linux vmware.
- ❖ I created a Test.txt file in A.1 to copy to A.2. To do this we:

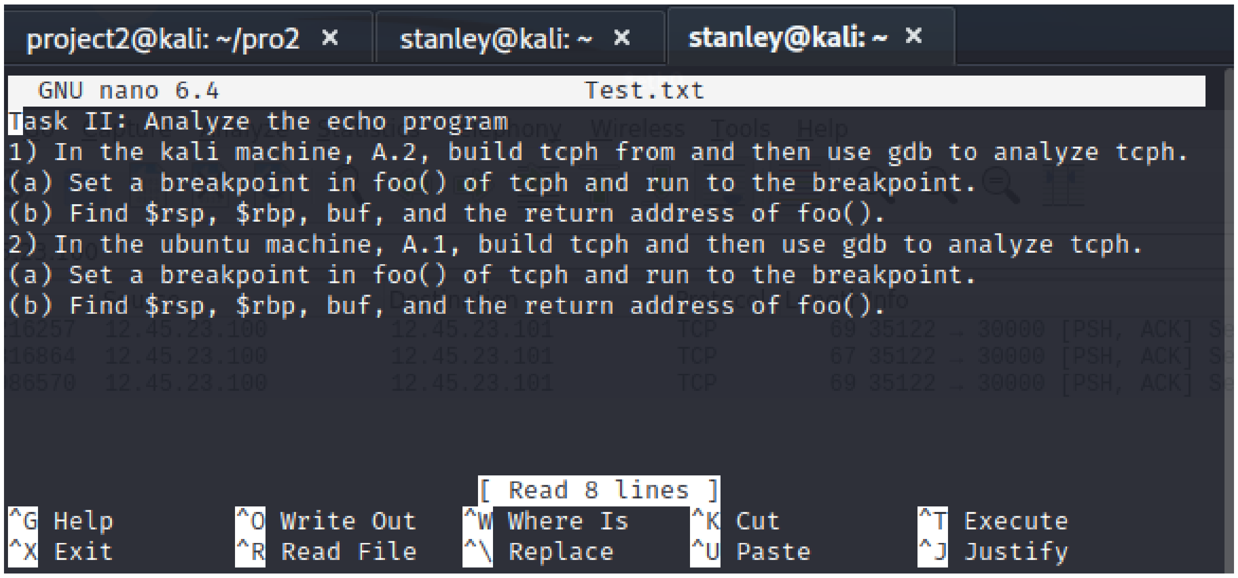
  scp username@A.1.ip.address:/path/to/files/* .

  Ans: scp stanley@12.45.23.101:/home/stanley/Documents/TS/* .



```
┌──(stanley㉿kali)-[~]
└─$ scp stanley@12.45.23.101:/home/stanley/Documents/TS/* .
stanley@12.45.23.101's password:
Test.txt                                          100%  439   189.1KB/s   00:00
```

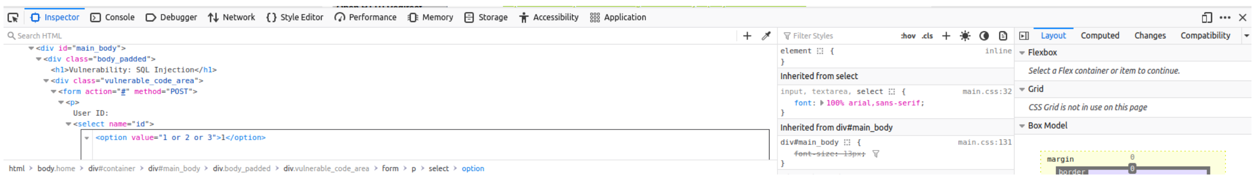
(iv) Show the content of the smallest file in the retrieved files.



```
project2@kali: ~/pro2 ×    stanley@kali: ~ ×    stanley@kali: ~ ×
 GNU nano 6.4                          Test.txt
Task II: Analyze the echo program
1) In the kali machine, A.2, build tcph from and then use gdb to analyze tcph.
(a) Set a breakpoint in foo() of tcph and run to the breakpoint.
(b) Find $rsp, $rbp, buf, and the return address of foo().
2) In the ubuntu machine, A.1, build tcph and then use gdb to analyze tcph.
(a) Set a breakpoint in foo() of tcph and run to the breakpoint.
(b) Find $rsp, $rbp, buf, and the return address of foo().




                                [ Read 8 lines ]
^G Help          ^O Write Out     ^W Where Is      ^K Cut           ^T Execute
^X Exit          ^R Read File     ^\ Replace       ^U Paste         ^J Justify
```

(v) Show the injected SQL statement.

- ❖ The injected SQL statement added in the injection source file was done with "or" to confuse the database:

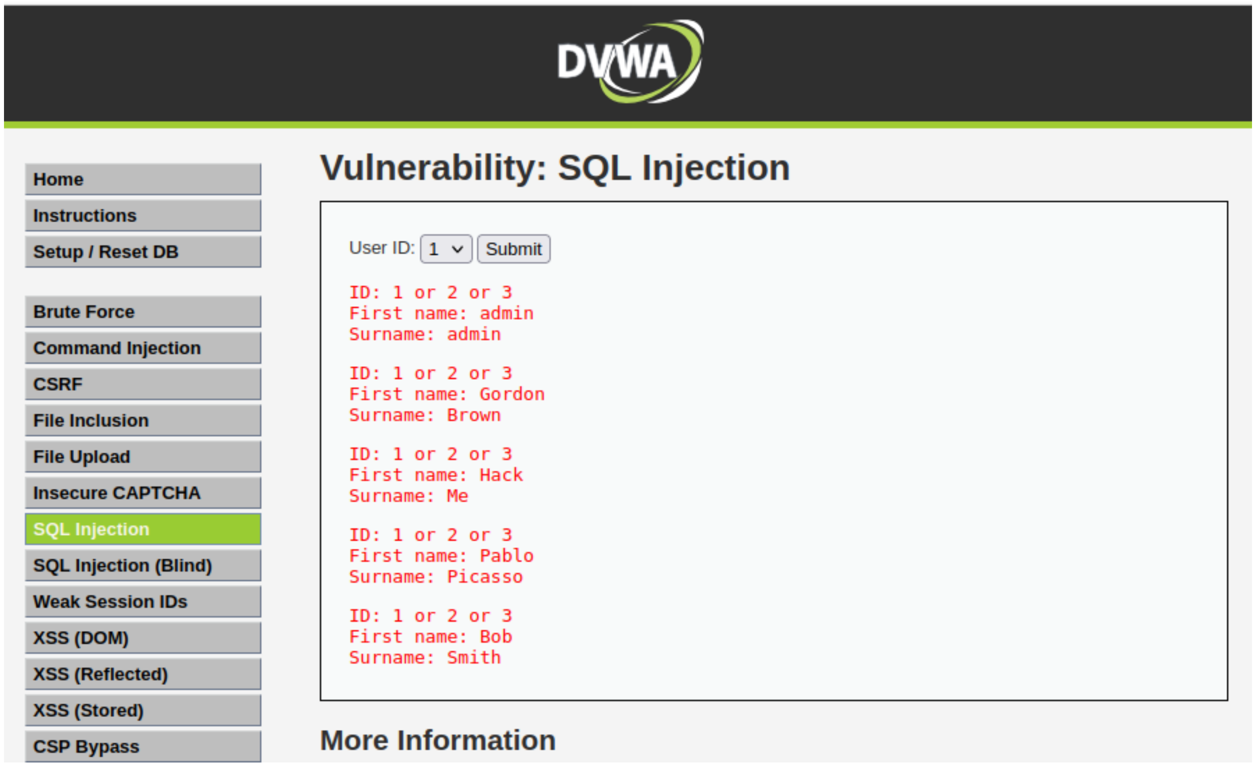


```
<div id="main_body">
  <div class="body_padded">
    <h1>Vulnerability: SQL Injection</h1>
    <div class="vulnerable_code_area">
      <form action="#" method="POST">
        <p>
          User ID:
          <select name="id">
            <option value="1 or 2 or 3">1</option>
html > body.home > div#container > div#main_body > div.body_padded > div.vulnerable_code_area > form > p > select > option
```

```
element {                                   inline
}
Inherited from select
input, textarea, select {               main.css:32
  font: 100% arial,sans-serif;
}
Inherited from div#main_body
div#main_body {                        main.css:131
  ~~font-size: 13px;~~
}
```

Flexbox
Select a Flex container or item to continue.
Grid
CSS Grid is not in use on this page
Box Model
margin 0
border 0

(vi) Show the screenshot of the web page that show all user IDs, first names, and last names.



Vulnerability: SQL Injection

```
ID: 1 or 2 or 3
First name: admin
Surname: admin

ID: 1 or 2 or 3
First name: Gordon
Surname: Brown

ID: 1 or 2 or 3
First name: Hack
Surname: Me

ID: 1 or 2 or 3
First name: Pablo
Surname: Picasso

ID: 1 or 2 or 3
First name: Bob
Surname: Smith
```

More Information

Section V:

(i) One defense mechanism is to randomize the address space of stack memory (so called randomization). The shell scripts, enablerandom.sh and disablerandom.sh, are provided to show how to enable or disable the defense mechanism.
To enable the defense mechanism, run the following command in the terminal:
./enablerandom.sh
To disable the defense mechanism, run the following command in the terminal:
./disablerandom.sh

(ii) Discuss the reason that randomization can defeat the attack.

By relocating the address space in a program's memory, randomization makes it challenging for attackers to determine the location of code present in the stack memory. This, in turn, makes it difficult for them to discover an address and predict the location of the stack memory. If attackers know the code's location, they won't be able to uncover the data's location, which they need to exploit vulnerabilities. Hence, randomization helps to prevent unauthorized access to data.

Also, address space randomization can also prevent buffer overflow attacks. In a buffer overflow attack an attacker can send more data than a program expects, which causes the program to overwrite important memory addresses. With address space randomization, the attacker cannot predict where the overflow will occur, which makes it much more challenging for the attacker to exploit.

(iii) 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 will it take for the attacker to compromise the server?

- If only the low 16 bits of the stack address are randomized, there are 2^16 possible addresses that the stack can be located at.
- Assuming an attacker sends 10 exploiting packets every second, the probability that one of those packets will hit the correct address is 1/2^16, or approximately 0.0015%.
- The expected number of packets needed to successfully compromise the server is 1/0.0015%, or approximately 66,667 packets.
- Therefore, it would take approximately 6,666.7 seconds, or 111 minutes and 7 seconds, for the attacker to compromise the server.