Saishnu Ramesh Kumar (300758706)

CSCI 191T - Assignment 2 (Shellshock and Buffer-Overflow)

**Section 1:**

1. No, the command will not execute. The vulnerability can only exist when the variable of the value starts with (){}. Otherwise, this would be considered a regular string and foo will become a variable in the child process.

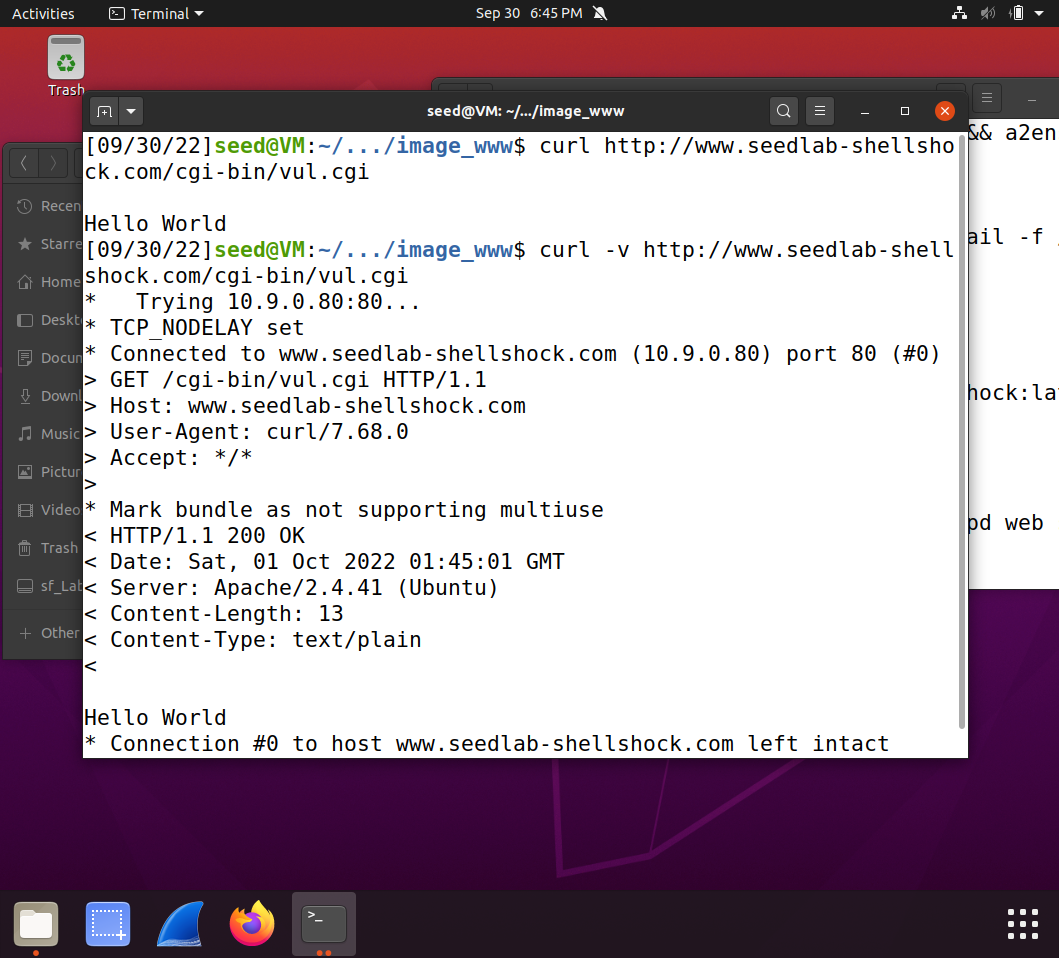
1. When fork() is introduced into the program, it divided the system into two parts, the parent and child procedures. Fork() would return the child processes ID to the parent process and will return 0 to the child. Therefore, the child PID keeps the ID of the child process within the fork (). From observing the program, the output was parent and child in new lines each. The reason for there not being any other output is that there is a NULL in the execve which causes the program to not output anything else as it checks through the entire program and goes to the parent procedure then loops back into the child procedure to NULL.

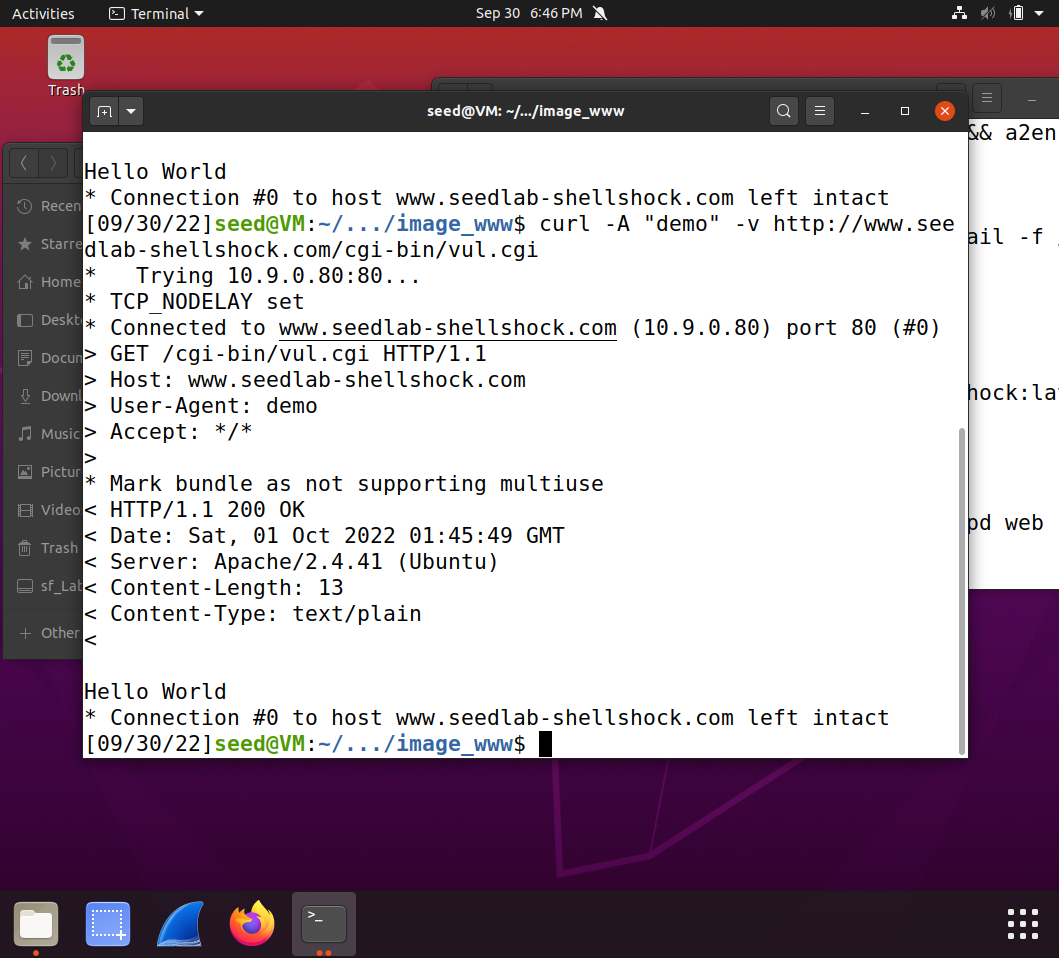
Graphical user interface, text

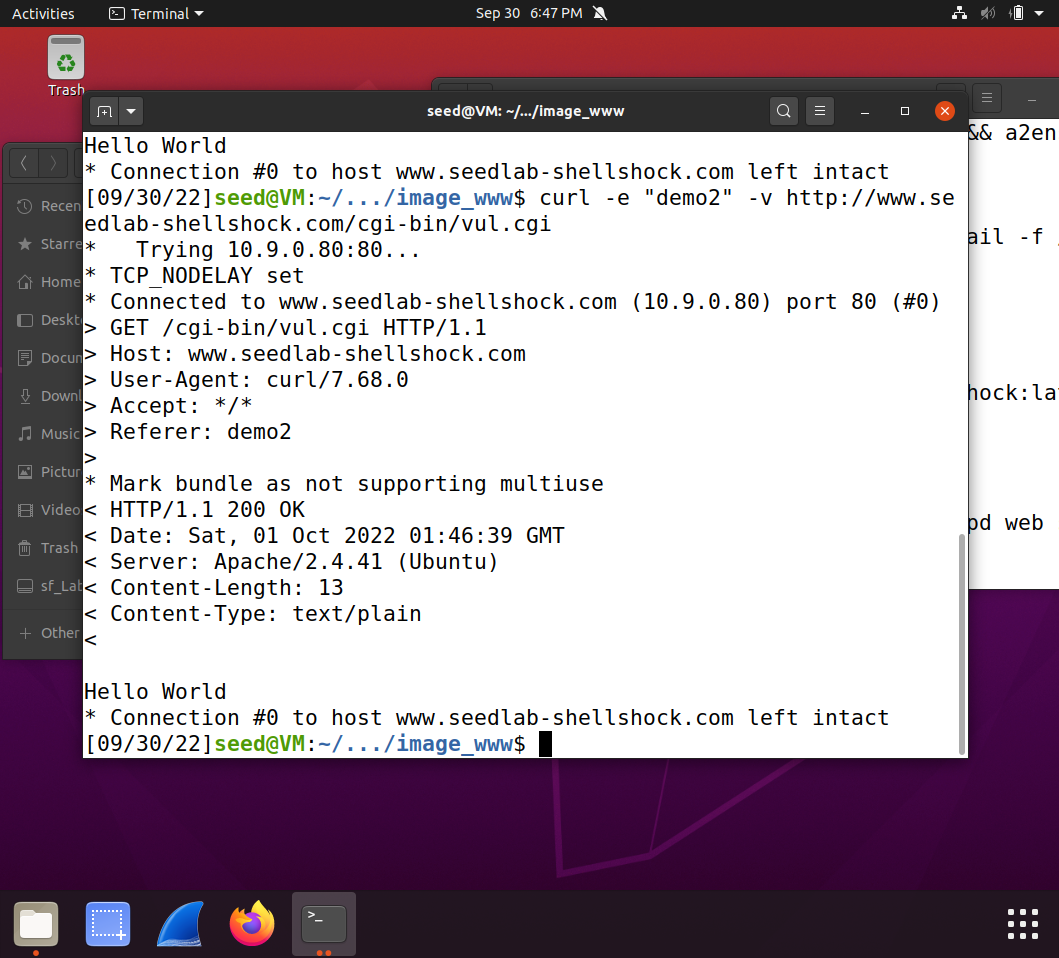
Description automatically generated

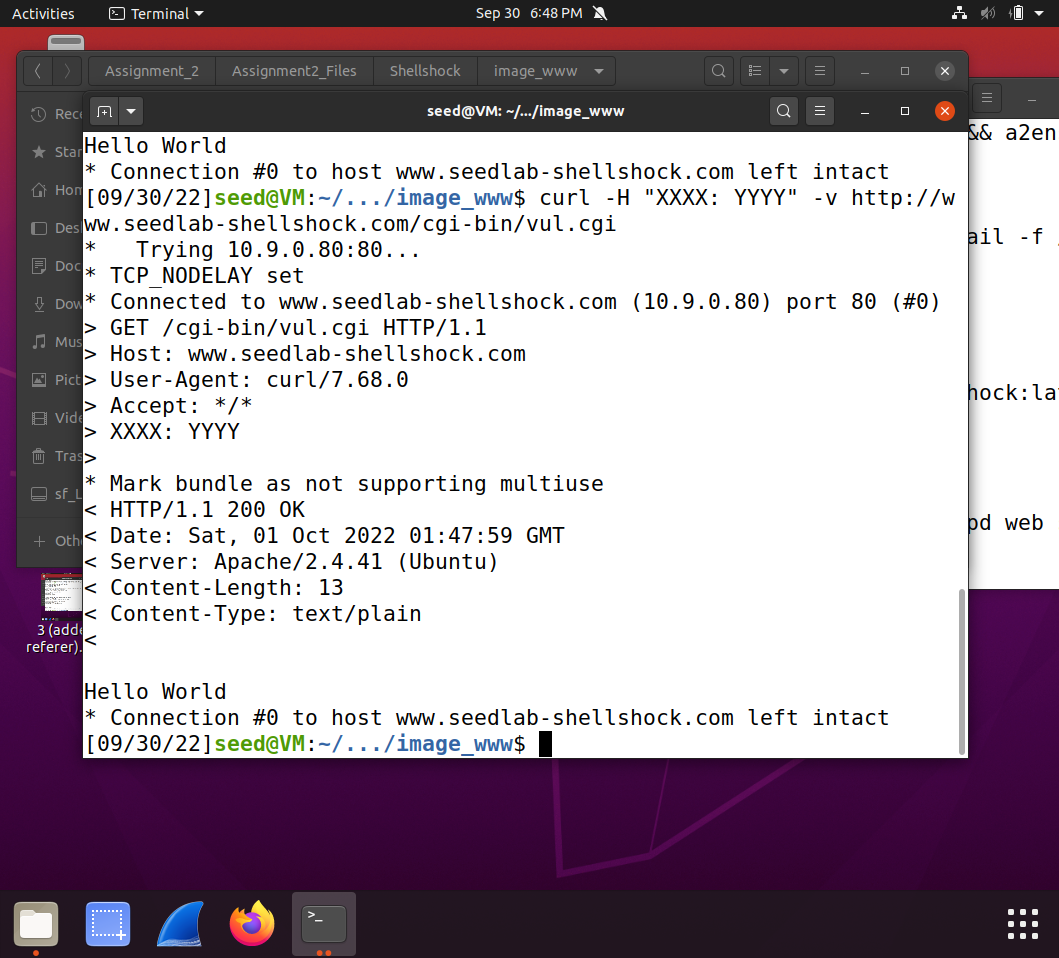
1. The variable int i is a global variable and is initialized with the integer 0 therefore, it is in the initialized data segment. The str is a parameter of the function and is in the stack. The ptr variable is also located in the stack but it points to a memory location that is dynamically allocated and can be found in the heap. The variable j is a local variable, so it is in the stack. The buf variable as it is a local array, it is found in the heap. Finally, as for y, since this is a static variable, it is not initialized therefore it is in the uninitialized data segment.
2. For a 32-bit system (which is 4 bytes), we can fill the string up with a double word (of 32 bits). Since characters are 1 byte, we can request the buffer with 24 + 1 bytes = 25 bytes and the system would be able to allocate 7 double words (28 bytes). Therefore, we would be able to essentially spray the return address across the buffer but leaving some space for the NOPs to allow the attack to take place. Therefore, when we input more than 28 bytes, it causes a buffer overflow. In conclusion, we can input a string of 28 characters or less to allow some space for the NOPs between the input code and attack.
3. In this situation, we are running in a 64-bit system (which is 8 bytes) and we are also able to fill this up with double words of 64 bits. From 0 to 64 bytes, we will be able to allocate some empty space and add NOPs as well as the return address before the program goes into the malicious code segment of the program. We can also add a string offset into the buffer. In the code, strcpy is used to copy the string and if the 24 bytes and value is true, it will return 1 otherwise it will be false. Therefore, in between 64 and 67, we would then be able to add 4 bytes of the malicious code in between those bits to create an attack on the system.

**Section 2 (Shellshock Attack):**

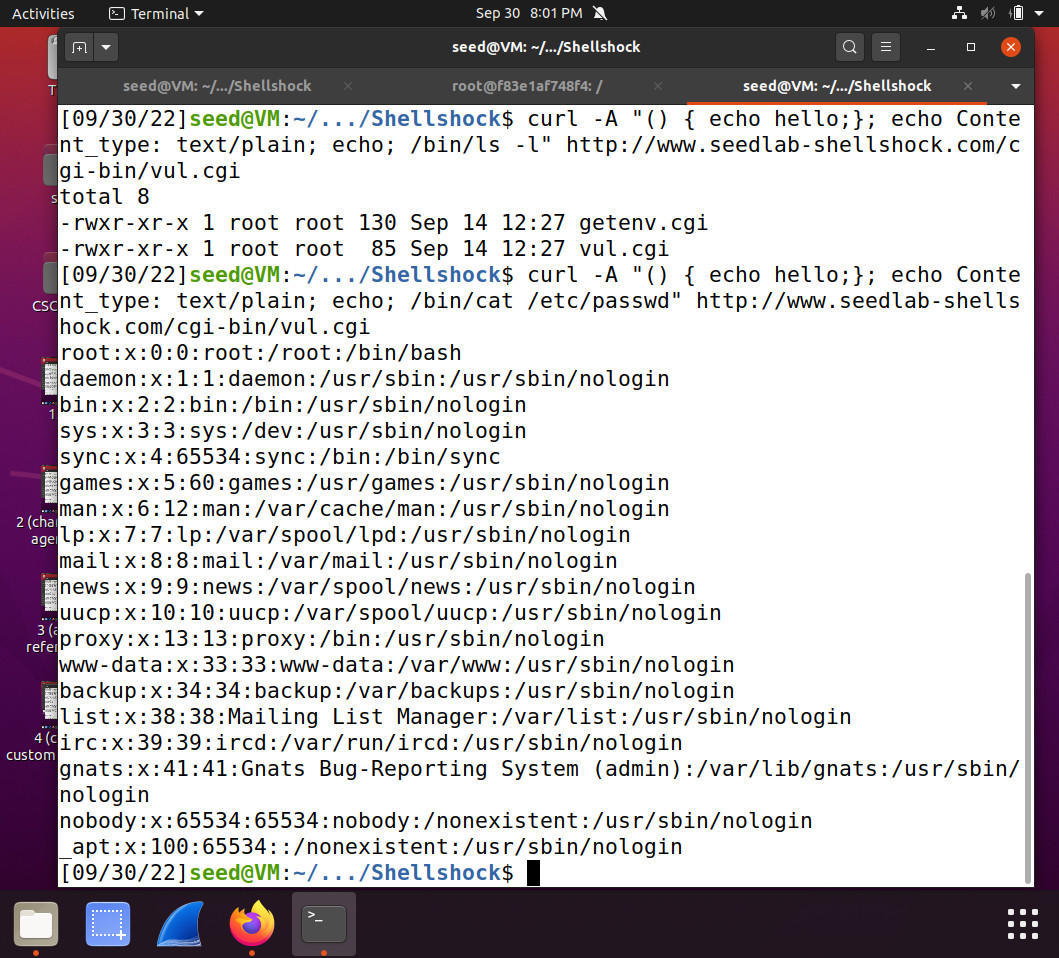
1. -v: This was the default output for this section. It was able to gain the information from the victim server. 

-A: By using the -A field, we can change the name of the user agent, as seen below. The new name of the user-agent is demo. 

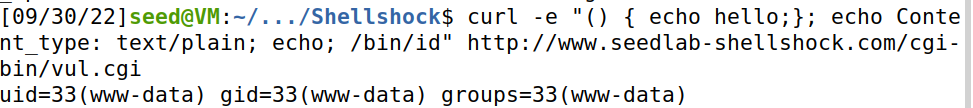
-e: When we use the -e field, this keeps the original user-agent name but instead it adds and changes the referer name, demo2, as in the default output, there was no referer.

-H: Finally, for the -H section, this creates a custom referer. As seen below, the referer is now known as XXXX and the name assigned to it is YYYY because with -H we can add two variables into the program. 

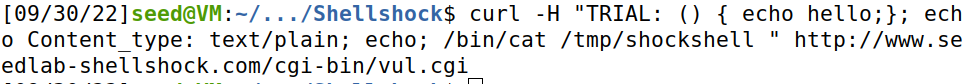
1. For Q1 (curl -A): When we run this, we are able to gain access to all the passwords found within the system.

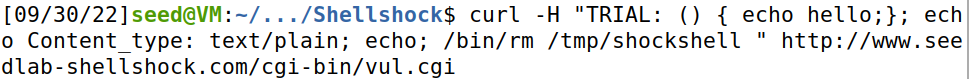


For Q2 (curl -e): When running this part, we can get the UID by using /bin/id.

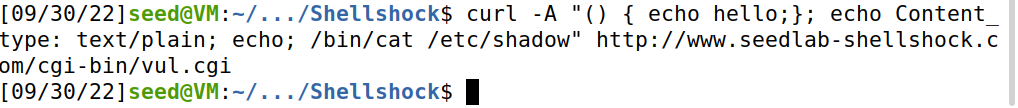


For Q3 (curl -H): This would add the file shockshell into the tmp folder.



For Q4 (curl -H): Using /bin/rm removes the shockshell file from the tmp folder.

1. Question 1: No, you will not be able to steal the contents from the shadow file because the file is a privileged file therefore no one can have access to it except the owner. As seen in the screenshot when we try to access it, the terminal outputs nothing.



Question 2: When entering that code and URL into the terminal, we get the environment variables and, in the REQUEST\_URI, we get the AAAAA that was entered at the end of the URL and the QUERY\_STRING outputs AAAAA. But when attempting to run the Shellshock, it does not work and as seen in the second screenshot below (the first one states all the environment variables found), it states 400 Bad Request. Therefore, to conclude, this method is unable to launch the Shellshock attack.

Graphical user interface, text, application

Description automatically generated

