

A PROJECT REPORT

ON

**LINUX CAPABILITY EXPLORATION**

SUBMITTED TO

**Professor JUZI ZHAO**

(Instructor – CMPE-209 Section-2)

By

**Project Team**

**SAI SREE VAISHNAVI CHITTOORI (012415130)**

**PRIYANKA SUBRAHAMANYAM(012470835)**

Department of Computer Engineering

November 17, 2017

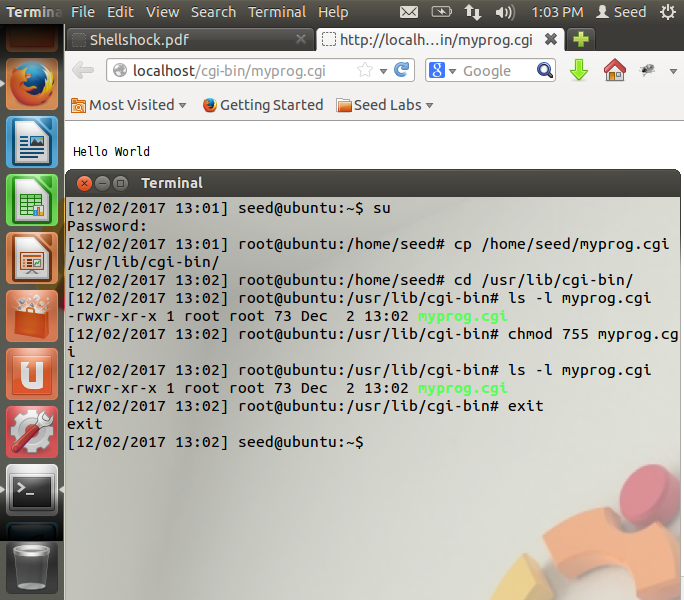
**CONTENTS**

|  |  |
| --- | --- |
| **Topic** | **Page No.** |
| Task 1: Experiencing Capabilities | 3 |
| Question 1 Explore non-set UID programs |  |
| Question 2 Explore Capabilities |  |
| Task 2: Adjusting Privileges | 4 |
| Install enable/disable/delete functionality for capability | 6 |
| Question3 cap\_dac\_read\_search capability adjustment | 7 |
| Question 4 Capability vs ACL | 8 |
| Question 5 Capability vs Buffer-Overflow | 13 |
| Question 6 Capability vs Race Condition Attack | 13 |
| Conclusion |  |
| **Topic** | **Page No.** |
| Task 1: Experiencing Capabilities | 3 |
| Question 1 Explore non-set UID programs |  |
| Question 2 Explore Capabilities |  |
| Task 2: Adjusting Privileges | 4 |
| Install enable/disable/delete functionality for capability | 6 |
| Question3 cap\_dac\_read\_search capability adjustment | 7 |
| Question 4 Capability vs ACL | 8 |
| Question 5 Capability vs Buffer-Overflow | 13 |
| Question 6 Capability vs Race Condition Attack | 13 |
| Conclusion |  |

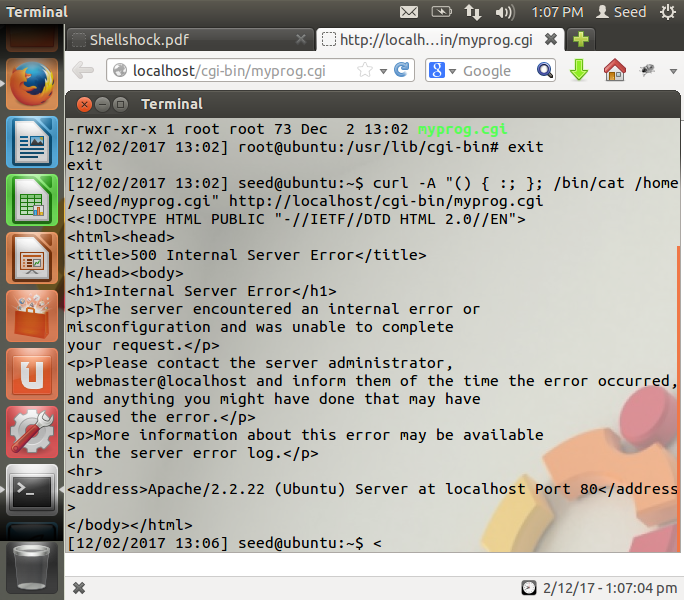
|  |  |
| --- | --- |
| **Topic** | **Page No.** |
| Task 1: Experiencing Capabilities | 3 |
| Question 1 Explore non-set UID programs |  |
| Question 2 Explore Capabilities |  |
| Task 2: Adjusting Privileges | 4 |
| Install enable/disable/delete functionality for capability | 6 |
| Question3 cap\_dac\_read\_search capability adjustment | 7 |
| Question 4 Capability vs ACL | 8 |
| Question 5 Capability vs Buffer-Overflow | 13 |
| Question 6 Capability vs Race Condition Attack | 13 |
| Conclusion |  |
| **Topic** | **Page No.** |
| Task 1: Experiencing Capabilities | 3 |
| Question 1 Explore non-set UID programs |  |
| Question 2 Explore Capabilities |  |
| Task 2: Adjusting Privileges | 4 |
| Install enable/disable/delete functionality for capability | 6 |
| Question3 cap\_dac\_read\_search capability adjustment | 7 |
| Question 4 Capability vs ACL | 8 |
| Question 5 Capability vs Buffer-Overflow | 13 |
| Question 6 Capability vs Race Condition Attack | 13 |
| Conclusion |  |

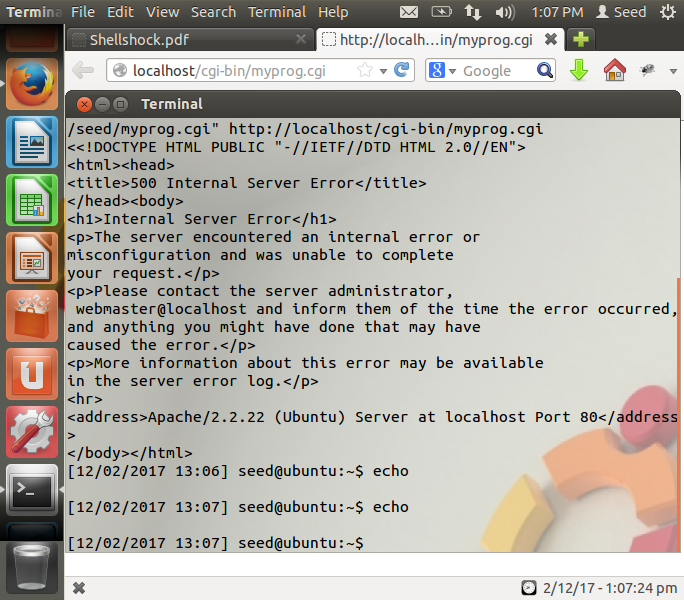
**Shellshock Attack Lab Report**

**Task 1 Procedure:**

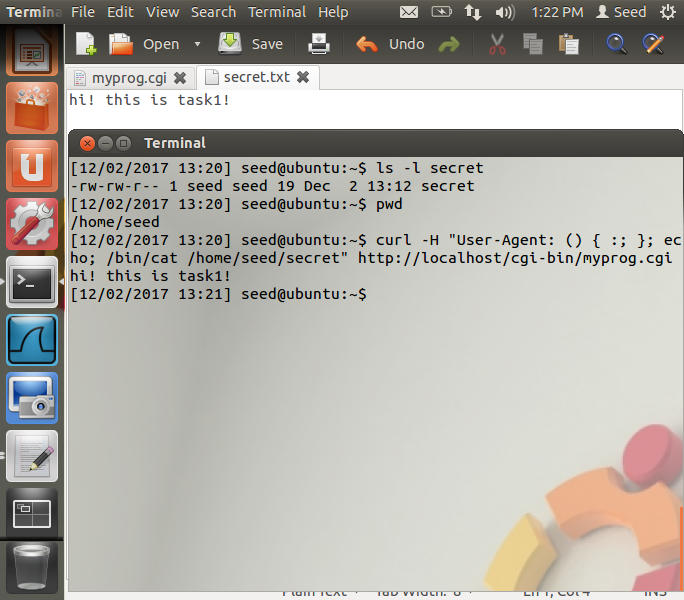
****

**Figure 1.1**

****

****

**Figure 1.2**

****

Attack is successful as we can see that the file is printed to he shell.

**Figure 1.3**

**Procedure:**

1. **In the terminal start by entering su mode.**
2. **Then enter root and run the .cgi file as shown in the above screen shots.**
3. **Here we use the curl command to print output of the Common Gateway Interface (.cgi) program.**
4. **Through this, we make the .cgi file executable. Place the file in the /usr/lib/cgi-bin folder.**
5. **In step 2 ,we are able to get a file that cannot be accessed by the attacker and print the contents of that file.This is due to the success of the attack.**
6. **To perform this operation we enter the below command in the console:-**

**curl -H "User-Agent: () { :; }; echo; /bin/cat /home/seed/secret"** [**http://localhost/cgi-bin/myprog.cgi**](http://localhost/cgi-bin/myprog.cgi)**,**

**–H in the command allows us to processes the string and also performs the command program which is in our case the .cgi file.**

1. **There is a problem is in the *initialize\_shell\_variables(env, privmode) in* variables.c file**

* ***if (privmode == 0 && read\_but\_dont\_execute == 0 && STREQN ("() {", string, 4))***

If the privilege mode is not the root, the attack will fail. So always check for Privilege.

***{…***

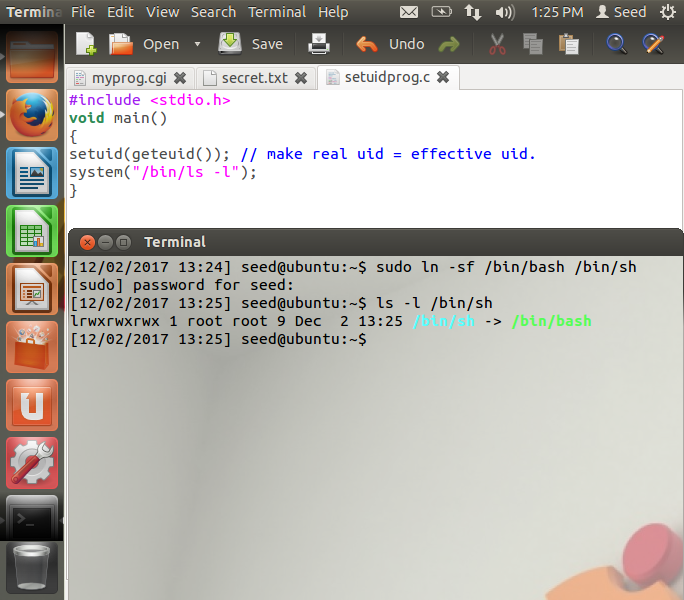
***}***

**Given that an environment variable starts with a string ‘() {’ The function , initialize\_shell\_variables recognizes it as a function definition.**

***parse\_and\_execute (temp\_string, name, SEVAL\_NONINT|SEVAL\_NOHIST);***

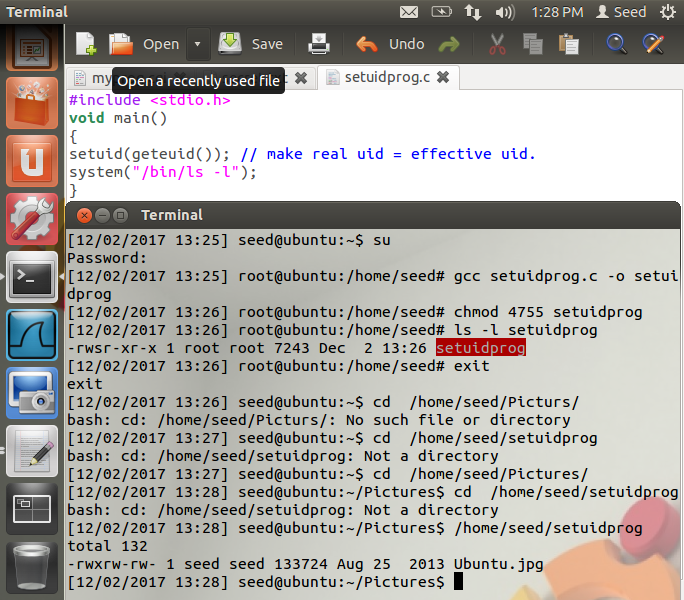
* **After this, the remaining portion of the string is passed to be parsed and executed.**
* **We are assuming that is assumed that the rest of the string is passed without providing the checks since it contains only the function execution.**
* ***parse\_and\_execute()* does not stop executing even after the end of the function. This is a huge problem.**
* **Since bash runs all the commands in the string, sometimes even some after the function definition is over, command injection is possible.**
* **This because an attacker can control an environment variable in a program that will create a shell with an environment containing that variable,**

**Task 2 Procedure:**

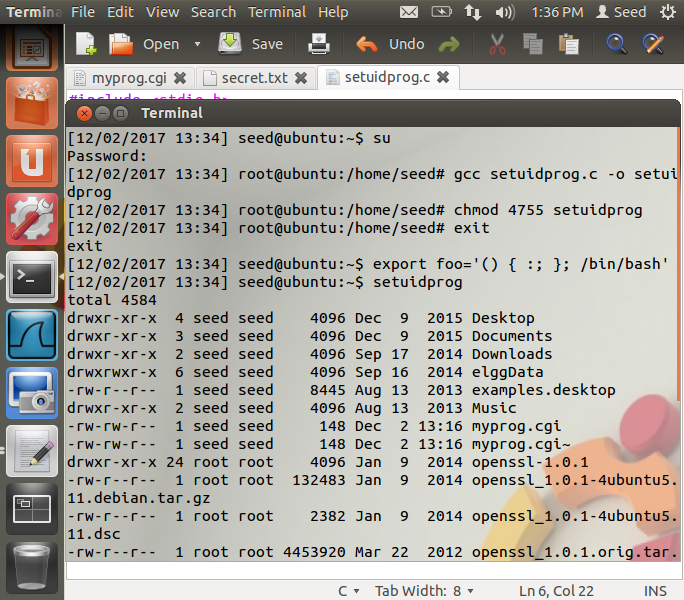
****

Link bash to sh

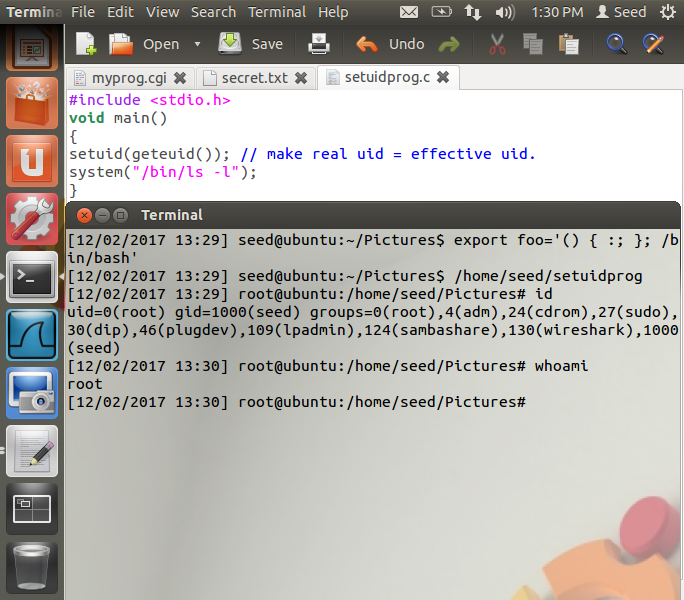
**Figure 2.1**

****

**Figure 2.2**

****

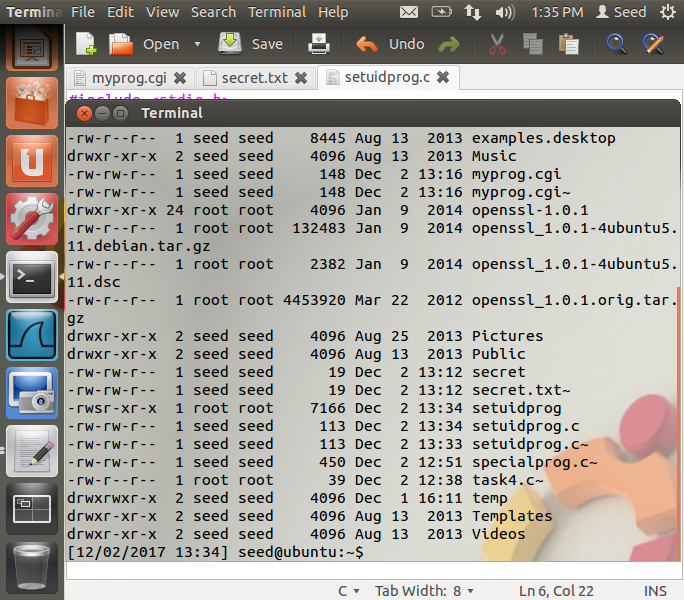
**Figure 2.3**

****

Successful! Root access gained

Setting Vulnerability

**Figure 2.4**

****

No root access->Fail

**Procedure:**

1. **The first two diagrams we create the basic setup that we need for this task.**
2. **Use “export foo=..” as shown in Figure 2.4 to set vulnerability.**
3. **In the figure 2.3 we compile *setuidprog.c* giving it root permissions. The programs used are shown above in screenshot.**
4. **Since we export the vulnerability causing sequence in the task, the injection causes the unavoidable execution of commands at the end of the function definitions, stored in the values of environment variables.**
5. **On removing the “ *setuid(geteuid()); // make real uid = effective uid.”* statement from the program as shown in the above diagram, we are unable to gain root access.**
6. **The main reason behind this is if the real user id and the effective user id are the same, the function defined in the environment variable is evaluated, and so the bash door vulnerability can be exploited.**
7. **But the real uid and effective uid are not the same which is the reason the attack fails.**

**Task 3:**

1. **Other than the two scenarios described above (CGI and Set-UID program), is there any other scenario that could be affected by the Shellshock attack? We will give you bonus points if you can identify a significantly different scenario and you have verified the attack using your own experiment.**

There are a few other scenarios where Shellshock can be used to exploit a system,

* IBM restricted shell- The IBM restricted shell can be overcome and the user can gain access to unrestricted shell using the same process that we’ve followed.
* SSH server- The ForceCommand feature in OpenSSH ,where instead of running an unrestricted shell , a fixed command is executed, even if the user specifies a special shell program to be run. If we put this command into the environment variable “ssh\_original\_command”. When this forced command is run as Bash shell, the bash shell will parse this environment variable on start up, and run the commands in it. The user has used their restricted shell access to gain unrestricted shell access, using the Shellshock bug.
* Email systems- A gmail mail server can pass external input through to bash, Depending on the system configuration, a that will enable exploitation of a vulnerable version of gmail. In fact, on 6th October, it was reported that Yahoo! Servers were compromised as a result of a variation of the shellshock attack.
* DHCP- DHCP can pass commands to Bash, when it is connected to an open Wi-Fi network. A DHCP client can request and fetch more than the IP address from a DHCP server, but can also be provided a series of additional options. Strings can be sent to local machines for execution.

1. **What is the fundamental problem of the Shellshock vulnerability? What can we learn from this vulnerability?**

* In general, shellshock lets an attacker execute arbitrary code on web servers.
* Bash lets users define functions as a way to pass text on to other systems and processes.
* The difficulty with this vulnerability, which includes specific characters as part of a definition i.e like “ () { :; }; “ is that doesn’t stop processing the string containing the function after it is defined, it will continue to read and execute shell commands following the function definition.
* This will allow the attacker to get access to the shell. This doesn’t mean that the attacker has gotten root.
* It simply allows the attacker to continue the attack and attempt privilege escalation, which might lead to access to root.
* This vulnerability shows the importance of sanitation of strings being sent to a shell.
* The parser’s problem is that even if there is code after the end of the function declaration, that code will be inadvertently be executed.