Project 1: Introduction to Penetration Testing

Fall 2021

The goal of this project :

Penetration testing is an important part of ensuring the security of a system. This project provides an introduction to some of the common tools used in penetration testing, while also exploring common vulnerabilities (such as Shellshock and setUID bit exploits).

On September 24, 2014, a severe vulnerability in Bash, nicknamed Shellshock, was identified. This vulnerability can exploit many systems and be launched either remotely or from a local machine. In this project, you will gain a better understanding of the Shellshock vulnerability by exploiting it to attack a machine. The learning objective of this project is to get first-hand experience on this interesting attack, understand how it works, and think about the lessons that we can get out of this attack.

Environment Setup :

Files Provided	Description
shellshock_server.ova	The VM image (see setup Step 18).
assignment_questionnaire.txt	Template for final submission.

This project requires the use of virtual box and multiple VMs.

Installing Virtual Box:

1. Install VirtualBox if it is not already installed. This project requires the latest version of VirtualBox 6.1.x. Using an earlier version of Virtual Box has been known to cause errors where the project VM freezes, so if you run into this, ensure you're running on at least Virtual Box 6.1.0 or later.

2. Download the Oracle Virtual Box Extension Pack (available for download at the same location as Virtual Box is).

3. Import the Extension Pack under File-> Preferences-> Extensions

4. In Virtual Box go to Tools -> Preferences -> Network.

5. Select the small plus sign in the upper right corner of the network preferences box to create a new NAT network. Select the NAT network and select the small gear below the add button to edit it. In the box that appears, change the name to something related to the project. Then save it, and close the preferences.

Installing the Kali Linux VM:

6. Visit https://images.kali.org/virtual-images/kali-linux-2021.2-virtualbox-amd64.ova . This will begin an auto-download of the Kali Linux VirtualBox VM.

7. Alternatively (if the above link does not work), visit: https://kali.org/get-kali/ and select “VIrtual Machines” and then “Virtual Box”.

8. Import the Kali Linux VM to Virtual Box. Once you’ve imported the VM, you'll likely need to go into the VM Settings and increase the number of CPUs if possible, as well as the RAM. Also enable 3-d acceleration, and set the zoom level to 300 (it makes it easier to read).

9. Go to the new Kali VM’s settings. In the network tab change “Attached to:” from NAT to NAT Network in the Adapter 1 tab. The name of the network should autofill to your newly created network if you only have one, but if you have multiple NAT networks, you'll need to select the correct one.

10. Repeat the process with the other VMs you want to communicate with the Kali VM.

11. Start the Kali VM. The default username/password is: **kali**/**kali**.

12. Ensure the guest additions are installed. If they’re not, go to devices->insert guest additions CD). Then follow the instructions in the popup window to install the guest additions..

13. In the top Virtual box menu bar of the Kali VM, go to Devices -> Shared Folders -> Shared Folders Settings.

14. Click Machine Folders to ensure it's highlighted. Click the small add folder icon on the right of the screen.

15. Select a path (in "Folder Path") to a folder on your host machine that will act as the shared folder on the host. Give the folder a name, if it doesn't autofill already. Check the "Auto-mount" box. Check the "Make Permanent" box. Click okay, and you should now see the folder under "Machine Folders".

16. In a terminal on the Kali VM, run:

```
sudo usermod -a -G vboxsf $(whoami)
```

17. Now logout of the Kali VM and log back in (or just restart the Kali VM). The shared folder should be under: /media/sf_{shared folder name}

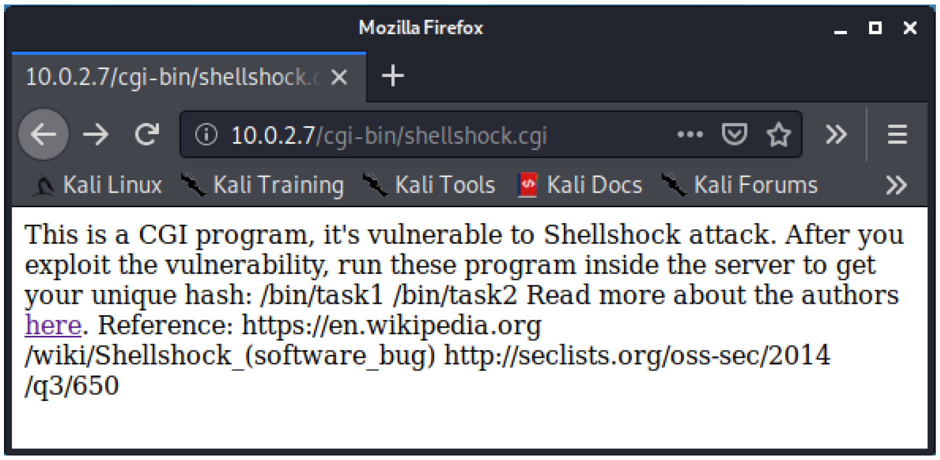
Vulnerable machine:

18. Download the project appliance via any of the following methods:
    1. https://drive.google.com/file/d/1vVnWkpXqyPMRRS5YI4zHrhYVq4Ia4OtN/view?usp=sharing sha256sum: 04557c059c367f3828f95a656f2f363c8577ce55d24e834b9c1b4ca4b24a529a

19. Import the downloaded OVA into VirtualBox (File -> Import Appliance). Check the VM’s Network setting. Ensure Adapter 1 is set to NAT Network and the name of the NAT Network is the name you specified when creating the NAT Network in step 5.

20. Try to connect to the VM server from the Kali. Start the VM in VirtualBox (as no login is needed, you can use headless start, accessible by selecting the "Headless Start" option under the "Start" drop

down menu in Virtualbox). Once you find the IP of the VM in Task 1 below, navigate to the following URL: http://<IP address of the shellshock_server VM>:<http port found in part 1>/cgi-bin/shellshock.cgi
Then you should be able to see the content:



This is a CGI program, it's vulnerable to Shellshock attack. After you exploit the vulnerability, run these program inside the server to get your unique hash: /bin/task1 /bin/task2 Read more about the authors here. Reference: https://en.wikipedia.org/wiki/Shellshock_(software_bug) http://seclists.org/oss-sec/2014/q3/650

21. Notice that you don't need the password to access the web content hosted on the VM server. Instead of using a real server, it's safer to perform the attack on an emulated Apache HTTP Server VM. To be clear, you will not be logging into the shellshock VM during this project. Once you configure the shellshock VM (by following steps 17-20 above), you'll be exploiting it externally, from the Kali VM, by exploiting the Shellshock vulnerability.

Project Tasks (100 points):

Task 1: Network Scanning – (20 points)

The first step in any penetration test is to gather information about the network and servers you'll be exploiting. In this task, you will perform network scanning and answer a few questions based on your findings. On startup, the shellshock VM listens to several ports for incoming messages.

1. Find the IP address of the vulnerable VM on the NAT network.
2. Find the port number of http traffic to the Apache web server on the VM.
3. Find the list of open ports in the VM whose service name is shown as "unknown". **Hint: nmap can be used to scan open ports. Make sure to scan all ports!** If you're finding only one open port with the name "unknown", you may not be using the right nmap command.
4. Establish connection with each of the open ports with service name "unknown" that you found in question 2. Once connection is established, the server receives a unique integer answer message. **Hint: netcat can be used to listen to a particular port.**

Task 2: Attack CGI Program– (20 points)

In this task, you will launch the Shellshock attack on a remote web server. Many web servers enable CGI, which is a standard method used to generate dynamic content on Web pages and Web applications. Many CGI programs are written using a shell script. Therefore, before a CGI program is executed, the shell program will be invoked first, and such an invocation can be triggered by a user from a remote computer.

To access this CGI program from the Web, you need to first check the server VM is running. Then, you can either use a browser by typing the following URL:

http://<IP address found in part 1>:<http port found in part 1>/cgi-bin/shellshock.cgi

or use the following command line program curl to do the same thing:

```
$ curl  http:// <IP address found in part 1>:<http port found in part 1>/cgi-bin/shellshock.cgi
```

For this task, your goal is to launch the attack through this URL, such that you can achieve something that you cannot do as a remote user. For example, you can execute some file on the server, or look up some file located on the server. When you successfully launch an attack, please execute the /bin/task2 program (which needs your GT username as the input) inside the VM. It will generate the submission hash for you.

For students that want to verify their work, here's an example correct input/output for /bin/task2:

```
$ /bin/task2 g1
Here is your task2 hash:
db449890172b7ebedd0243637a5071c86d6f3d750c4b5fbf3ebba7796b854df2
```

Task 3: Reverse Shell with Metasploit – (20 points)

Now you've successfully launched the Shellshock attack, and you can execute commands on the server VM. However, during a penetration test, you likely won't have time to craft a payload for every exploit you use. And, what if the server wasn't in fact vulnerable to shellshock. How would you know if your exploit failed because it was wrong, or because there wasn't a vulnerability?

That's where Metasploit comes in. Metasploit is a standard in the penetration testing community. It allows pen testers to run precompiled exploits against a host, using predefined payloads. For this project, we're going to use Metasploit to establish a reverse shell between your machine and the host machine.

Let's first see how Metasploit works by using it to scan the open tcp ports of the shellshock_server VM. While this is a task better suited to tools like nmap (as seen in Task 1), it serves as a good demonstration of how to use a Metasploit module. If you've used Metasploit before, you can skip this introduction and go directly to the Task 3 assignments section at the end.

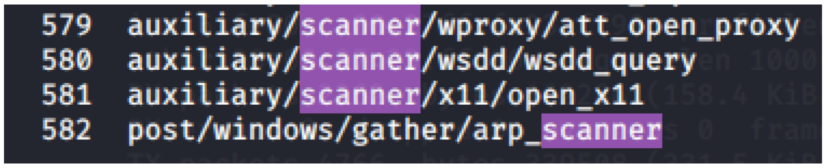
1. Begin by opening a new terminal on your Kali VM. In the new terminal type: `msfconsole`. After a moment, the Metasploit Framework console (msfconsole) will load. For this project, the msfconsole is the main way of accessing Metasploit. While there are other tools and command prompts associated with Metasploit, the msfconsole is suitable for the entirety of this project.

2. For this example, we're going to scan the ports of a host. You can use the results from task 1 to ensure Metasploit is behaving properly. In practice using a Metasploit module solely for the purpose of scanning ports on a host is a little overcomplicated, since nmap can do much more and takes less setup, but this does offer a good introduction to using a Metasploit module.

3. A Metasploit module is the base of any task performed in Metasploit. It roughly consists of Ruby code that is written to perform a certain task (like exploit a certain vulnerability, or scan a certain kind of system). There are multiple different varieties of modules, but for our purposes we'll focus on three of them:

a. **The Exploit modules:** These are modules written to exploit a certain vulnerability.
b. **The Auxiliary modules:** These are modules written to perform some task related to exploiting a system (like scanning). Within the auxiliary modules there are many different kinds of modules. We'll be using the "scanner" modules for this example.
c. **Payloads:** Calling these modules is a little misleading. They are the payloads (for example the shellcode) that are sent within the exploit.

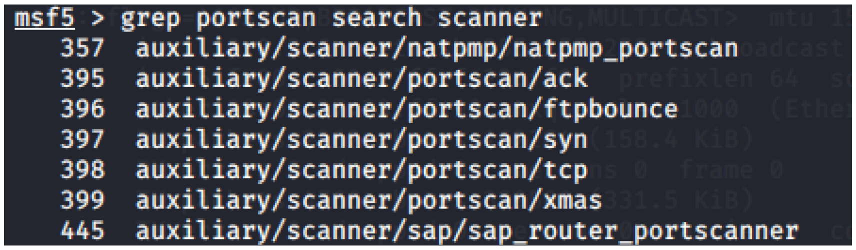
4. First, we have to find the correct module. We know it's scanning the system, so let's start by searching for "scanner" using the command (in the msfconsole) `search scanner`. Hmm... that's quite a few results (582 at the time of writing this).



```
  579  auxiliary/scanner/wproxy/att_open_proxy
  580  auxiliary/scanner/wsdd/wsdd_query
  581  auxiliary/scanner/x11/open_x11
  582  post/windows/gather/arp_scanner
```

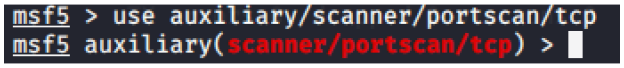
The reason there are so many results is that "scanner" is a class of auxiliary modules (as seen by there being so many modules beginning with "auxiliary/scanner"). So searching for "scanner" (or "scan") will give everything at all related to network or vulnerability scanning.

5. So, we need to narrow our search. One way of doing this is with the "grep" command, which filters the output. Since we're looking to scan the ports, lets try `grep portscan search scanner` to search for "portscan" within the search results from "scanner". We get:



```
msf5 > grep portscan search scanner
   357  auxiliary/scanner/natpmp/natpmp_portscan
   395  auxiliary/scanner/portscan/ack
   396  auxiliary/scanner/portscan/ftpbounce
   397  auxiliary/scanner/portscan/syn
   398  auxiliary/scanner/portscan/tcp
   399  auxiliary/scanner/portscan/xmas
   445  auxiliary/scanner/sap/sap_router_portscanner
```

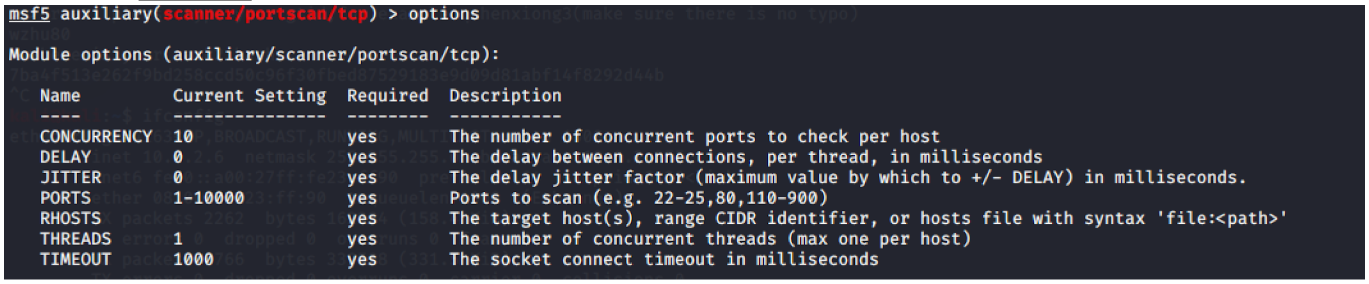
That seems a little better. Only 7 results. Since we're scanning for open tcp ports, let's use: "auxiliary/scanner/portscan/tcp". In order to use a metasploit module, you should run the command: `use module_name`



```
msf5 > use auxiliary/scanner/portscan/tcp
msf5 auxiliary(scanner/portscan/tcp) > 
```

You should now see "auxiliary(scanner/portscan/tcp)" after "msf5" indicating you are using the auxiliary(scanner/portscan/tcp) module.

6. Now that we're using the correct module, we have to set the correct options. Try running the command `options`. You should see something like:



```
msf5 auxiliary(scanner/portscan/tcp) > options

Module options (auxiliary/scanner/portscan/tcp):

   Name         Current Setting  Required  Description
   ----         ---------------  --------  -----------
   CONCURRENCY  10               yes       The number of concurrent ports to check per host
   DELAY        0                yes       The delay between connections, per thread, in milliseconds
   JITTER       0                yes       The delay jitter factor (maximum value by which to +/- DELAY) in milliseconds.
   PORTS        1-10000          yes       Ports to scan (e.g. 22-25,80,110-900)
   RHOSTS                        yes       The target host(s), range CIDR identifier, or hosts file with syntax 'file:<path>'
   THREADS      1                yes       The number of concurrent threads (max one per host)
   TIMEOUT      1000             yes       The socket connect timeout in milliseconds
```

While each of the options is marked as required, most of the pre-filled values work for our purposes. The only one we need to change is RHOSTS. RHOSTS should be the IP address of the host we want to scan. In this case, you should set it to the IP address of the shellshock_server VM, that you found in part 1. You can use the command: `set rhosts IP_of_host`. Now try running `options` again and ensure the RHOST option is set to the right IP address.

7. Now run `run` and you should see the same list of open ports that nmap showed. While "run" is usually used to run auxiliary exploits, the command "check" and "exploit" are often used to check and run exploit modules.

Now you've seen an example of how to use Metasploit. You'll follow a similar process when exploiting the shellshock vulnerability.

Task 3 Assignments:

1. Find an exploit module that exploits the shellshock vulnerability on an Apache web server. Once you've found the module, place the module name in assignment_questionnaire.txt.
2. Use `show payloads` to show the possible payloads for the module. Find a payload that spawns **a reverse tcp shell**. Place the full name of the payload in assignment_questionnaire.txt.
3. Run the exploit and spawn a reverse shell on the VM.
4. Run /bin/task3 in the resulting shell, then type cs6262 then your user ID. Report the hash value for your user ID in assignment_questionnaire.txt.

You'll submit all of your answers for this section in assignment_questionnaire.txt. You should keep the reverse shell running after finishing Task 3, as you will need it in Task 4.

Task 4: Privilege Escalation – (20 points)

Your goal:
You aim to upgrade the privilege for your command shell by exploiting the setUID vulnerability. You will run `\bin\task4` as the higher privileged user "shellshock_server", not the default user "www-data".

Background:
In Unix based systems, setUID is access rights flags that determine what users can run a program. For instance, when users want to change their password, they may run the `passwd` that requires root privilege. The setUID can help the user run the `passwd` with temporarily elevated privileges. However, if setUID is misused or setUID flags are misconfigured, it can cause a variety of vulnerabilities such as information leakage, unwanted privilege escalation, etc.

Assignments:
As a first step, type `whoami` in your shell. You should see "www-data" which is your user ID. Now, run `\bin\task4 your_userID`. You would see a permission denied error. That is because `\bin\task4` is configured to allow only the "shellshock_server" user to run it. So, you need to find a way to run the task4 as the "shellshock_server" user. A feasible approach is to spawn a shell running as a "shellshock_server" user, and run the task4 through it.

You goal is to find a program which:

1. Hash a higher privilege than the default user.
2. Can spawn a shell.

You may want to ransack `\usr\bin` for a program which has a higher privilege than the default user, and run `\bin\task4 your_userID` in the shell spawned. .

What is the vulnerable program? What command do you use to search it? What command do you use to spawn a shell with the vulnerable program? And what is the hash value from `\bin\task4 your_gtid` (similar to /bin/task2)? Please leave your answers in assignment_questionnaire.txt.

Task 5: Password Cracking - (20 points)

An invaluable part of any penetration test is password cracking. While there may be no known vulnerabilities in a system, a weak password could be just as damaging in allowing an attacker to gain access to a system (or view sensitive information once they gain access). We're going to look at two kinds of weak passwords in this task: passwords that are too short, and passwords that can easily be guessed via password scraping.

To begin, start a Meterpreter shell (using a meterpreter shell payload) through the Metasploit shellshock module in Task 3. A Meterpreter shell is different from the reverse TCP shell in Task 3, as it allows you to run metasploit specific commands on the vulnerable machine (like `download`). Navigate to `\home\shellshock_server\secret_files\`. There are two encrypted .pyc files here. `task51.zip` is encrypted with zip, while `task52.pyc.gpg` is encrypted with gpg (a common file encryption tool in Linux). Download these two files (`task51.zip` and `task52.pyc.gpg`) to your Kali VM using the meterpreter.

We already know the developers of this web server aren't very security savvy, since they let a shellshock vulnerability plus a setUID exploit give a high privilege shell on their machine. So, chances are they didn't pick very secure passwords for these secret files. Your goal in this task is to crack the passwords of these two files using John the Ripper (a popular password cracker) and cewl (a password scraper).

The command line tools used in Task 5 are located in `/usr/sbin` on the Kali VM. In order to run them, you can either add /usr/sbin to the $PATH variable, or write `/usr/sbin/` before each command.

You should use `zip2john` and `gpg2john` to extract the password hashes from the encrypted files. For `task51.zip`, try running John the Ripper incrementally. Report your John the Ripper command in assignment_questionnaire.txt (whether you also report your the zip2john and gpg2john commands is up to you, but they won't be graded).

For `task52.pyc.gpg`, try running John the Ripper incrementally again. Hmm… it seems to run forever. That's because John the Ripper is trying every possible combination of characters. If the password is too long (among other things), John the Ripper could run for years before it finds it.

But, just because the password is too long to be found incrementally, doesn't mean it can't be cracked. Take a look at the shellshock.cgi page in the browser. It looks like it gives a link to a profile of the authors. If the authors aren't great at picking secure passwords, maybe the password is something about them that we can guess from their profile page.

But, even if the password is on the profile page, it can still take a while to guess by hand. What if the password was kItt3n$ or deVEL0p3r. It would be hard to guess that, even if the word it was based on (like "kittens", or "developer") was on the profile page.

Instead, let's use a password scraper to create a custom wordlist for John the Ripper. For this project, we suggest using cewl. cewl is a relatively simple command line program that comes preinstalled on Kali, and creates password wordlists off of webpages. Since we want to get every possible password (including if the authors based the password off something on shellshock.cgi, or one of the landing pages), you should run cewl on shellshock.cgi, with the proper settings to ensure it follows the links all the way to profile.cgi (you may need to tune the cewl parameters). Report your cewl command in assignment_questionnaire.txt. Then try running John the Ripper on `task52.pyc.gpg` with the wordlist (and the default wordlist rules for testing different permutations of the words). Report your John the Ripper command in assignment_questionnaire.txt.

Once you find the passwords, report them in assignment_questionnaire.txt. Then decrypt the two files (using zip for `task51.zip` and gpg for `task52.pyc.gpg`) and run `python task51.pyc your_user_ID` and `python task52.pyc your_user_ID`. Report the resulting hash values for your user ID in assignment_questionnaire.txt.

Deliverables:

Please use **Gradescope** to submit the assignment files. The link to Gradescope is found in Canvas under Courses tab -> Gradescope. In Gradescope under active assignments click project 1 to upload your files.

- assignment_questionnaire.txt
- README.txt

The provided template "assignment_questionnaire.txt" marks where you should add your answers for each question. **Please DO NOT edit anything other than the fields marked for your answers (or student ID) in assignment_questionnaire.txt.** Doing so may result in the autograder failing to process your submission.

Please note that there is a 5-point penalty for not following the format given in assignment_questionnaire.txt.

You should also include a "README" plain text file explaining how and why each piece of your solution works. Including it makes grading easier if we cannot reproduce your results. You may also include screenshots in your submission to show proof. If you use any outside sources not mentioned in this write-up, then the README would be a good place to mention them as well. Although there is no specific format for the README, since it is not officially graded, we do offer a sample format under sample-README.txt.

Reminders:

Please submit the files EXACTLY as requested to Canvas. DO NOT package them up (e.g., as a ZIP file). Any deviations may result in a deduction of your grade.

There is a 5-point penalty for not following the submission format shown.

Useful Links and References:

- Shellshock Vulnerability
  - https://github.com/carter-yagemann/ShellShock
  - https://en.wikipedia.org/wiki/Shellshock_(software_bug)
  - http://seclists.org/oss-sec/2014/q3/650
- curl
  - https://curl.haxx.se/docs/manpage.html
  - https://curl.haxx.se/download.html (curl.exe for Windows)
- netcat
  - https://linux.die.net/man/1/nc
  - https://eternallybored.org/misc/netcat/ (nc.exe for Windows)
- nmap
  - https://nmap.org/book/man.html
- Metasploit
  - https://www.offensive-security.com/metasploit-unleashed/metasploit-fundamentals/
- John the Ripper
  - https://www.openwall.com/john/doc/
- cewl

- https://tools.kali.org/password-attacks/cewl

Acknowledgment:

This Lab was modified from SEED Labs, Copyright 2014 Wenliang Du, under the terms of GNU Free Documentation License, Version 1.2. The original document can be found at http://www.cis.syr.edu/~wedu/seed/

Checklist/Rubric:

Section		Points	✓
1	**Network Exploration**	**20**	
1.1	Correct first digits of the IP Address of the vulnerable VM.	5	
1.2	Correct HTTP port.	5	
1.3	Correct Port/Message Pairs (points divided evenly over all pairs)	10	
2	**Exploiting the System**	**20**	
2.2	Correct hash value from running /bin/task2.	20	
3	**Spawning a Shell with Metasploit**	**20**	
3.1	Correct exploit module	5	
3.2	Correct payload module (there are multiple correct answers here)	5	
3.3	Correct hash value from running /bin/task3.	10	
4	**Privilege Escalation**	**20**	
4.2	Correct vulnerable program name.	10	
4.4	Correct hash value from running /bin/task4	10	
5	**Password Cracking**	**20**	
5.2	Correct password for task51.zip.	5	
5.3	Correct hash value from running python task51.pyc.	5	
5.6	Correct password for task52.pyc.gpg.	5	
5.7	Correct hash value from running python task52.pyc.	5	
-	**Possible Deductions**		
i.	Incorrect assignment_questionnaire.txt format.	-5	
ii.	Incorrect upload to Gradescope/Canvas.	-5	
2.1	No command given for exploiting the shellshock vulnerability.	-5	
4.1	No command given for finding the vulnerable program.	-5	
4.3	No command given for exploiting the setUID vulnerability.	-5	
5.1	No command given for cracking task51.zip.	-5	
5.4	No command given for scraping the shellshock server webpage.	-5	
5.5	No command given for cracking task52.pyc.gpg.	-5	
+	Your Submission Includes Total:	100	
	assignment_questionnaire.txt – Answers to questions		
	README.txt		