Secure Programming
Path Traversal Attack Lab

In this lab, you will explore and demonstrate how an insecure file download function can be exploited using a **path traversal attack**. The Coffeeshop application does not have a component that is vulnerable to path traversal so for demonstration, we are adding a simple piece of functionality that downloads a file when we access the URL.

1. Create a directory called **files/** inside your project's root directory (inside the coffeeshop app). This is where the downloadable files will be stored. Path should be:

```
/vagrant/coffeeshopsite/coffeeshop/files/
```

2. Create a file named **secrets.txt** inside the files/ folder. Add some dummy content to it.
3. Verify the file has synced the /files folder to your container by using the following command (ssh into the container first):

```
ls /vagrant/coffeeshopsite/coffeeshop/files/
```

4. On the host, open your views.py file in a code editor and insert the **download_file** function (you can find it in the download_file.py file on Brightspace). NOTE: be sure to keep the indentation the same as Python has strict rules on indentation. The function takes a file parameter from the URL and attempts to open the secrets.txt file from the /files directory. The path construction is insecure, allowing a user to potentially traverse directories outside of the intended folder.
5. Add the following to your urls.py code at the end of the url_patterns list:

```
path('download_file/', views.download_file, name='download_file'),
```

It should look something like the following:

```
path('api/', include(router.urls)),
path('api-auth/', include('rest_framework.urls', namespace='rest_framework')),
path('download_file/', views.download_file, name='download_file'),
```

6. On your container, restart your Apache server:

```
sudo service apache2 restart
```

7. Now on your host, open the following:

8. http://localhost:8080/download_file/?file=secrets.txt. All going well, you should be able to download the secrets.txt file from the folder you created earlier.

Performing the Attack

1. We are going to try to gain unauthorised access to the **config.env** folder because it contains all the username/ passwords for the site. Realistically, such a file should not exist, but if it did, it should have strict access controls set. The config.env file is located in the secrets folder (outside the vagrant folder on your host)

```
\django-coffeeshop\coffeeshop\secrets
```

When you accessed the secrets.txt file, the application fetched the file from: **\coffeeshop\vagrant\coffeeshopsite\coffeeshop\files**. The objective is to now traverse out of the folder and up to the **config.env** file. We do this using the dot-dot-slash notation.

Using the URL for the file download, start inserting ../ into the URL, just after the ?file= part, e.g.,

```
http://localhost:8080/download_file/?file=../secrets/config.env
```

Q1. What happens when you try to access the config.env file using this URL?

Keep incrementally adding ../ to the URL. You should eventually reach the point where the application responds with the config.env.

Q2. What was the final URL that successfully carried out the dot-dot-slash attack?

Fixing the Path Traversal Vulnerability

To fix this vulnerability, we need to fix the code so that a user cannot force browse out of the location using the ../ notation. The first security fix we will apply involves normalising the path to the secrets.txt file. We do this first by replacing the line of code:

```
file_path = os.path.join(base_directory, file_name)
```

with:

```
file_path = os.path.normpath(os.path.join(base_directory, file_name))
```

Explanation
The function **os.path.normpath** is the part that normalises the path to the file. It does this by cleaning up the path and resolves any . (current directory) or .. (parent directory) references, effectively flattening the path and removing any traversal attempts. So, for example, the **path ../../../secrets/config.env** will be resolved to **secrets/config.env.**

After normalisation, the line:

```
if not file_path.startswith(base_directory):
```

checks whether the final **file_path** starts with the **base_directory** (/vagrant/coffeeshopsite/coffeeshop/files/). It ensures that the user-provided file path, after normalisation, remains within the intended directory (i.e., files/ directory). If the normalized path starts with base_directory, it means the file is located inside the files/ directory, and access is allowed. If not, the file is outside the permitted directory, and access is denied. Restart the Apache server and try to run the dot-dot-slash attack again.

Q3. What is the result?