# Forewords

The project site is based on the starter code provided on Github at https://github.com/cybersecuritybase/cybersecuritybase-project. The project can be run with the NetBeans as were the earlier projects during the cyber security base course.

# 1. CSRF Attack

The following CSRF attack test is made using Burp Suite free edition v.1.7.15 (toolkit for web application security testing). Because this tool was used earlier during the course the tutorial of using it is not discussed here. The support and information about the tool can be found at portswigger.net. We assume that the project is running at localhost:8080

Issue: CSRF Attack

Steps to reproduce:

1. Open Burp Suite

2. Open Proxy tab and intercept the data traffic against the project site

3. Open browser & navigate to the project site localhost:8080/form

4. Enter user details to the form Name: test & Address: test

5. Press Submit

6. Burp Suite shows that no CSRF token is sent with the user data (the template project does this with a token)

7. The project site does not verify that the user actually intended to submit the desired requests

The CSRF token is needed to secure the application form against Cross Site Request Forgery. In this type of attack the attacker may get the victim user to execute actions of the attacker's choosing (in this case the attacker might get the information if the victim has signed up to the event).

The template source project using Spring framework has CSRF protection and CSRF tokens enabled by default. This is however possible to disable with parameter: "http.csrf().disable();" in Java Configuration. This kind of configuration can be a result of poor security implementation, malicious coder or just irresponsible developer who just left it there.

With Spring framework the fix is simple because of the default CSRF protection. In order to restore the protection the parameter: "http.csrf().disable();" has to be removed from the SecurityConfiguration.java.

# 2. Security Misconfiguration

This Attack is made using Mozilla Firefox 50.1.0. We assume that the project is running at localhost:8080

Issue: Security Misconfiguration

Steps to reproduce:

1. Open Mozilla Firefox

2. Navigate to the project site localhost:8080/form

3. Click ErrorPage link

4. You received misconfigured error page that reveals sensitive data that evil hacker can exploit

5. You know now that the admin page is localhost:8080/admin

The project has security misconfiguration. There is unnecessary link that leads to an erropage where an evil attacker can collect sensitive information about the web application and the structure of it. This kind of vulnerability can for example be due to legacy code, bad coding or jus irresponsible development. Error handling is one of the sections that hackers can misuse and needs to be handled correctly without providing any unnecessary information. Periodic scans and audits are needed to help detect this kind of vulnerabilities. Fix for this vulnerability is an ErrorPage with removed sensitive data.

# 3. Missing Function Level Access Control

This Attack is made using Mozilla Firefox 50.1.0. We assume that the project is running at localhost:8080

Issue: Missing Function Level Access Control

Steps to reproduce:

1. Open Mozilla Firefox

2. Navigate to the project site localhost:8080

3. Change the url from "localhost:8080/form" to " localhost:8080/admin"

4. You can now access to a page only for administrators and view sensitive data

The project has a flaw with missing function level access control which an evil attacker can use to grant access to view sensitive data. This flaw can be fixed for example by authenticating the admins that are trying to view the /admin page, or change the logic of viewing posted data and remove the admin-page.

# 4. Sensitive Data Exposure

This Attack is made using Mozilla Firefox 50.1.0. We assume that the project is running at localhost:8080 The attack goes along with the previous one. When going to the /admin page you can see all the information in clear text that users have sent to the application form.

Issue: Sensitive Data Exposure

Steps to reproduce:

1. Open Mozilla Firefox

2. Navigate to the project site localhost:8080/form

3. Enter following information to textboxes:

Name: TestName

Address: TestAddress

4. Press Submit

5. Change the url from "localhost:8080/form" to " localhost:8080/admin"

4. You can now view the information that have been sent to the application form.

If the project application would collect accurate address information with names the data could be misused by malicious users. Fixing this can be done by removing the line

<span th:text="${user.address}">address</span>

From the /admin page, so we only print the names who have signed to the event.

The sensitive data needs better protection that can be implemented for example with encryption and proper function level access control.

# 5. Unvalidated Redirect

The following unvalidated redirect attack test is made using Burp Suite free edition v.1.7.15 (toolkit for web application security testing) and Mozilla Firefox 50.1.0. Because this tool was used earlier during the course the tutorial of using it is not discussed here. The support and information about the tool can be found at portswigger.net

Issue: Unvalidated Redirect

Steps to reproduce:

1. Open Burp Suite

2. Open Proxy tab and intercept the data traffic against the project site

3. Open browser & navigate to the project site localhost:8080/form

4. Press link ErrorPage

5. Press button Go to Google

6. The intercepted traffic shows that the requested GET parameter is /redirect?url=www.google.com

7. Copy the parameter to create a complete url with an unvalidated redirect

8. Turn the Intercept off from the Burp Suite

9. Navigate to url localhost:8080/redirect?url=www.youtube.com/watch?v=dQw4w9WgXcQ

10.You have now been rickrolled via the project site

11. The project site accepted unvalidated url redirect

This attack was made by changing the url that was suppose to redirect a user to Google.com. Even the purpose seemed to be harmless this kind of vulnerability can be easily misused by a malicious user who crafts an unvalidated url and uses the project site to trick victims to malicious site. Preventing this kind of vulnerabilities can be simply done for example by avoiding using redirects and forwards or if used the user parameters should not be involved in calculating the destination. For example the fix could be a direct link added to the error.html page that lead to google <a href="http://www.google.com">Go to Google</a>

# 