WebApplication Security Lab

Lab showing several potential web application vulnerabilities and how to avoid them.

This example has been created for a Internet Application Programming (KIV/PIA) labs at the Department of Computer Science,
University of West Bohemia in Pilsen, Czech Republic.

Not validating User Input

Invalidated user input may lead to many security vulnerabilites.

Redirect/Forward URL

Allows phising attacks on the application user - by providing "special" URLs, users may be redirected to unwanted pages.

- Check the Route servlet in the project. Run the application and try to access the following URL: `http://localhost:8080/example-app-spring/route?where=http://zcu.cz`
- Such URL may be sent e.g. by email and user may easily miss it is actually malicious

Protection

- do not use the actual URLs as input parameters. Have e.g. a set of location names ("login", "homepage", "account") and let the application to map those values to the actual URLs.

Injection

Allows attacker to execute malicious code in your application. E.g. SQL.

- Check the UserDaoJPA.authenticate method - it uses direct parameter appending without validation.
- Try logging in as any (existing) user and the following password: `' OR 1=1--'`

Protection

- always escape your input parameters
- e.g. PreparedStatement parameter mechanism in JDBC

Cross-Site Request Forgery (CSRF)

An attacker takes advantage of you being logged into a page to send his own requests there on your behalf (e.g. transfer money from your bank account to his).

- Check the SecretMoneyServlet in the webapp package.
- Start the application and login as any user.
- Open the csrfattack.html in the src/main/webapp folder in the same browser as the one you used to log into the application.
- Click the Win Money button.
- Check system console for logs from SercretMoneyServlet
- Note that even the button clicking could be made automatically via JavaScript, hence you might be completely unaware of what happened.

Protection

- Send additional unique token with each request to validate next request's origin (malicious pages can trick your browser into sending cookies with login information, but they dont have access to your application data)
- The token should be valid for limited time
- Use only PUT, POST, PATCH and DELETE http methods to modify application state

Now let's try a solution in Spring Security:

- Go to applicationContext.xml and change `<security:csrf disabled=true/>` to `<security:csrf/>`
- Check register.jsp and index.jsp forms and their `<sec:csrfInput/>`

Configuration

- Wrong server configuration may put the whole application security useless. What if the server allows browsing of all files in the system? -> The attacker can e.g. download all database files, stored certificates, etc.
- Default username and passwords
- Leaving debugging options on in production (display system details to the attacker)

Cross-Site Scripting (XSS)

An attacker attempts to execute his own javascript in your browser.

- Login into the application and go to the secret/vip page
- Check the SecretServlet - the actual harmful code is part of the application data - could be inserted e.g. as a comment on the page and stored in the database.
- Or it could be part of a link (URL)
- Can be used to send your cookie to the attacker.
- See the XSS Details link in the **Some Reading** section.

Protection

- input validation and escaping (ensure whatever input user gives, it is never executed when displayed in the browser)
- HttpOnly header on the cookie - prevent's client-side code to access the cookie value. Optional feature, browsers must support it.

Some Reading

Top 10 Vulnerabilities by OWASP
XSS Details

License