Sahil Bhirud - 801138029

ITIS 5221

Preventing Injection Attacks

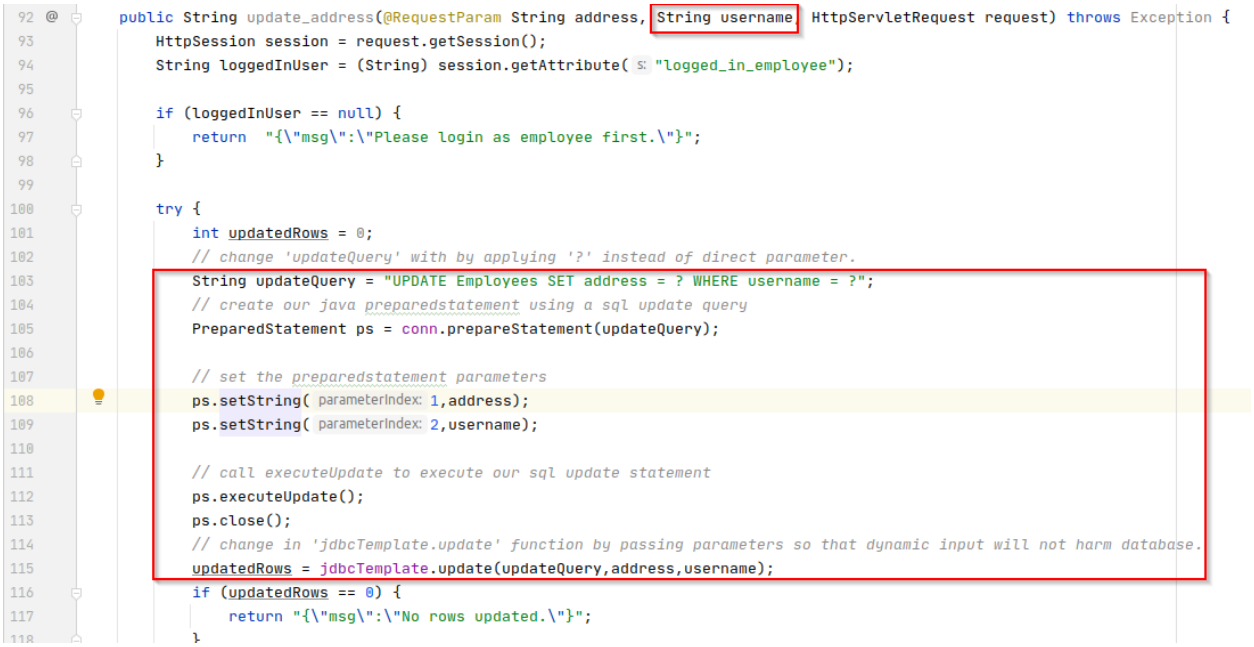
SQL Injection:

Original Code:

```
@ResponseBody
public String update_address(@RequestParam String address, HttpServletRequest request) throws Exception {
    HttpSession session = request.getSession();
    String loggedInUser = (String) session.getAttribute( s: "logged_in_employee");

    if (loggedInUser == null) {
        return  "{\"msg\":\"Please login as employee first.\"}";
    }

    try {
        int updatedRows = 0;
        // change 'updateQuery' with by applying '?' instead of direct parameter.
        String updateQuery = "UPDATE Employees SET address = '" + address + "' WHERE username = '" + loggedInUser + "'";
        // change in 'jdbcTemplate.update' function by passing parameters so that dynamic input will not harm database.
        updatedRows = jdbcTemplate.update(updateQuery);
        if (updatedRows == 0) {
            return "{\"msg\":\"No rows updated.\"}";
        }
        // change to display only logged-in employee's data
        String selectQuery = "SELECT * From Employees";
        List<Map> listOfEmployee = (List<Map>) findDataFromDatabase(selectQuery,  param1: null);
        return new ObjectMapper().writeValueAsString(listOfEmployee);
    } catch (Exception e){
        e.printStackTrace();
        return "{\"msg\":\"No rows updated.\"}";
    }
}
// ---------------------------------------------- Update address(2nd tab) ----------------------------------------
```

Changes made:

```
16  import java.sql.PreparedStatement;
17  import java.sql.Connection;

21  public class Sql_injectionController {
22
23      @Autowired
24      private JdbcTemplate jdbcTemplate;
25      public Connection conn;
```



```
92 @    public String update_address(@RequestParam String address, String username, HttpServletRequest request) throws Exception {
93          HttpSession session = request.getSession();
94          String loggedInUser = (String) session.getAttribute( s: "logged_in_employee");
95
96          if (loggedInUser == null) {
97              return  "{\"msg\":\"Please login as employee first.\"}";
98          }
99
100         try {
101             int updatedRows = 0;
102             // change 'updateQuery' with by applying '?' instead of direct parameter.
103             String updateQuery = "UPDATE Employees SET address = ? WHERE username = ?";
104             // create our java preparedstatement using a sql update query
105             PreparedStatement ps = conn.prepareStatement(updateQuery);
106
107             // set the preparedstatement parameters
108             ps.setString( parameterIndex: 1,address);
109             ps.setString( parameterIndex: 2,username);
110
111             // call executeUpdate to execute our sql update statement
112             ps.executeUpdate();
113             ps.close();
114             // change in 'jdbcTemplate.update' function by passing parameters so that dynamic input will not harm database.
115             updatedRows = jdbcTemplate.update(updateQuery,address,username);
116             if (updatedRows == 0) {
117                 return "{\"msg\":\"No rows updated.\"}";
118             }
```

The added code inserts placeholders (?) whose values are being set by calling the corresponding setter methods of the PreparedStatement thereby preventing the attack of SQL injection.

XSS Prevention:

Original Code:

```java
public class XssController {

    @GetMapping("/")
    public String xss_index() { return "xss/index"; }

    @GetMapping("/body_xss")
    @ResponseBody
    public String body_xss(@RequestParam String tagVal) throws Exception {
        return tagVal;
    }

    @PostMapping("/textarea_xss")
    @ResponseBody
    public Object textarea_xss(@RequestParam String tagVal) throws Exception {
        return tagVal;
    }

    @PostMapping("/js_xss")
    @ResponseBody
    public Object js_xss(@RequestParam String tagVal) throws Exception {
        return tagVal;
    }
```

Changes made:

```java
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.*;

import org.apache.commons.lang.StringEscapeUtils;
```

```java
public class XssController {

    @GetMapping("/")
    public String xss_index() { return "xss/index"; }

    @GetMapping("/body_xss")
    @ResponseBody
    public String body_xss(@RequestParam String tagVal) throws Exception {
        String encoded_html = StringEscapeUtils.escapeHtml(tagVal);
        return encoded_html;
    }

    @PostMapping("/textarea_xss")
    @ResponseBody
    public Object textarea_xss(@RequestParam String tagVal) throws Exception {
        String encoded_html = StringEscapeUtils.escapeHtml(tagVal);
        return encoded_html;
    }

    @PostMapping("/js_xss")
    @ResponseBody
    public Object js_xss(@RequestParam String tagVal) throws Exception {
        String encoded_html = StringEscapeUtils.escapeHtml(tagVal);
        return encoded_html;
    }

```

The escapeHtml() takes raw string as parameter and then escapes the characters using HTML entities.

Command Injection:

Original Code:

```java
    public Object command_injected(@RequestParam String command) {
        Map<String, String> response_data = new HashMap<~>();
        try {
            String output = "";
            String[] cmd = {"/bin/bash", "-c", "ping -c 3 " + command};
            Process p = Runtime.getRuntime().exec(cmd);
            p.waitFor();
```

Changes made:

```java
public Object command_injected(@RequestParam String command) {
    Map<String, String> response_data = new HashMap<~>();
    try {
        String output = "";
        //String[] cmd = {"/bin/bash", "-c", "ping -c 3 " + command};
        //Process p = Runtime.getRuntime().exec(cmd);
        //p.waitFor();
        ProcessBuilder processBuilder = new ProcessBuilder();
        processBuilder.command("ping","-c","3",command);
        Process p = processBuilder.start();
        p.waitFor();
```

The ProcessBuilder is used to create operating system processes. The start() method creates a new Process instance with the attributes and it also lets us control the environment.

Path Manipulation:

Original Code:

```java
public Map<String, String> view_file(@RequestParam String filePath) throws Exception {
    Map<String, String> response_data = new HashMap<~>();

    try {
        Resource resource = resourceLoader.getResource( s: "classpath:files/" + filePath);
        File file = resource.getFile();
        String text = new String(Files.readAllBytes(file.toPath()));
        response_data.put("status", "success");
        response_data.put("msg", text);
        return response_data;
    } catch (IOException e) {
        e.printStackTrace();
        response_data.put("status", "error");
        response_data.put("msg", "No output found");
        return response_data;
    }
```

Changes made:

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

```java
public Map<String, String> view_file(@RequestParam String filePath) throws Exception {
    Map<String, String> response_data = new HashMap<~>();

    String regex = "^[\\w,\\s-]+\\.[A-Za-z]{3,10}$";
    Pattern r = Pattern.compile(regex);
    Matcher m = r.matcher(filePath);
    if (m.find()) {
        try {
            Resource resource = resourceLoader.getResource( s: "classpath:files/" + filePath);
            File file = resource.getFile();
            String text = new String(Files.readAllBytes(file.toPath()));
            response_data.put("status", "success");
            response_data.put("msg", text);
            return response_data;
        } catch (IOException e) {
            e.printStackTrace();
            response_data.put("status", "error");
            response_data.put("msg", "No output found");
            return response_data;
        }
    } else {
        response_data.put("status", "error");
        response_data.put("msg", "Invalid Filename");
        return response_data;
    }
}
```

Here, the input data is sanitized using regular expressions.

Log Forgery/Injection:

Original code:

```java
        } catch (Exception e) {
            logger.info("After exception: " + encodedString);
            response_data.put("status", "error");
            response_data.put("msg", "Successfully logged error");
            return response_data;
        }
```

Changes made:

```java
    } catch (Exception e) {
        try {
            logger.info("After exception: " + java.net.URLEncoder.encode(encodedString,StandardCharsets.UTF_8.name()));
        } catch (UnsupportedEncodingException unsupportedEncodingException) {
            unsupportedEncodingException.printStackTrace();
        }
        response_data.put("status", "error");
        response_data.put("msg", "Successfully logged error");
        return response_data;
    }
}
```

The URL is encoded before using it so that the attack of log forgery can be avoided.

SMTP:

Original Code:

```
26
27          String name = customer_firstName;
28          String replyto = customer_email;
29          String message = customer_comments;
30          String to = "root@localhost";
31          String subject = "My Subject";
```

Changes made:

```
11  import java.util.regex.Matcher;
12  import java.util.regex.Pattern;
```

```
28  public String smtp_header_submit(@RequestParam String customer_firstName,@RequestParam String customer_email,@RequestParam String customer_comments) {
29
30      String SafeString = "^[.\\p{Alnum}\\p{Space}]{0,1024}$";
31      String emailPattern = "[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\\.[a-zA-Z]{2,}";
32      Pattern namePattern = Pattern.compile(SafeString);
33      Pattern emailPat = Pattern.compile(emailPattern);
34
35      Matcher nameMatch = namePattern.matcher(customer_firstName);
36      Matcher emailMatch = emailPat.matcher(customer_email);
37
38      if (nameMatch.find() && emailMatch.find()) {
39          String name = customer_firstName;
40          String replyto = customer_email;
41          String message = customer_comments;
42          String to = "root@localhost";
43          String subject = "My Subject";
44
45          String headers = "From:" + name + "\\n" + " to:" + replyto + "\\n";
46          String[] split = headers.split( regex: "\\\\n");
47          String y = "";
48          for (int i = 0; i < split.length; i++) {
49              y += split[i];
50              y += "<br>";
51          }
52          System.out.println(y);
53          return y + " Message:" + message;
54      }
55      else {
56          return "Invalid firstname or email";
57      }
58  }
```

In this case too, regular expressions are used to sanitize the data input to avoid attacks.

XPath:

Original Code:

```
59          XPathExpression expre = xpath.compile( expression: "/customers/customer[email = '" + val + "']/id/text()");
```

Changes made:

1. Created a new class SimpleVariableResolver:

XPath_injectionController.java | SimpleVariableResolver.java

```java
package net.uncc.app.xpath_injection;



import javax.xml.namespace.QName;

import javax.xml.xpath.XPathVariableResolver;

import java.util.HashMap;

import java.util.Map;



public class SimpleVariableResolver implements XPathVariableResolver {

    private final Map<QName, Object> vars = new HashMap<QName, Object>();

    public void addVariable(QName name, Object value) {

        vars.put(name, value);

    }



    public Object resolveVariable(QName variableName) {

        return vars.get(variableName);

    }

}
```

2. Imported necessary package(s):

```java
import javax.xml.namespace.QName;
```

3. Used the functions of SimpleVariableResolver to implement separation of domain:

```java
public Object xpath_injected(@RequestParam String val) {
    Map<String, String> response_data = new HashMap<~>();
    try {
        DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
        factory.setNamespaceAware(true);
        DocumentBuilder builder = factory.newDocumentBuilder();
        Document doc = builder.parse( uri: "src/main/resources/files/customer.xml");

        XPathFactory xpathFactory = XPathFactory.newInstance();
        XPath xpath = xpathFactory.newXPath();

        List<String> names = new ArrayList<>();
        //XPathExpression expre = xpath.compile("/customers/customer[email = '" + val + "']/id/text()");
        SimpleVariableResolver resolver = new SimpleVariableResolver();

        resolver.addVariable(new QName( namespaceURI: null,  localPart: "email_val"), val);

        xpath.setXPathVariableResolver(resolver);

        XPathExpression expre = xpath.compile( expression: "/customers/customer[email = $email_val]/id/text()");

        NodeList nodes = (NodeList) expre.evaluate(doc, XPathConstants.NODESET);
        for (int i = 0; i < nodes.getLength(); i++)
            names.add(nodes.item(i).getNodeValue());

        response_data.put("status", "success");
        response_data.put("msg", Arrays.toString(names.toArray()));
        return response_data;
    } catch (Exception e) {
        e.printStackTrace();
        response_data.put("status", "error");
        response_data.put("msg", "Successfully logged error");
        return response_data;
    }
```