BẢNG THEO DÕI SỬA ĐỔI

STT	Trang	Nội dung sửa đổi	Ngày có hiệu lực
1.	2,6	Bổ sung mục 2.1: *Cập nhật thư viện (Lib) bản vá cho web framwork:*	
2.	4,34	Bổ sung mục 2.18: *Các lỗi liên quan đến xử lý luồng nghiệp vụ, logic.*	
3	3,26	Bỏ lỗi Session Hijacking. Lý do: Là kỹ thuật tấn công, không phải là lỗi.	

1. Mục đích

- Đưa ra các yêu cầu về lập trình an toàn trong phát triển ứng dụng web.

2. Đơn vị áp dụng và tổ chức thực hiện

- Quy định này áp dụng với các Phòng, Ban Tập đoàn, các đơn vị hạch toán phụ thuộc Tập đoàn và các công ty hạch toán độc lập do Tập đoàn sở hữu từ 50% vốn điều lệ trở lên.
- Các Tổng Công ty, Công ty phụ thuộc và Công ty độc lập căn cứ vào quy định của Guideline này xây dựng quy định cụ thể hoá cho phù hợp với thực tế tại đơn vị.

3. Yêu cầu

2.1 Cập nhật thư viện (Lib) bản vá cho web framwork:

2.2 Tương tác với cơ sở dữ liệu tránh lỗ hổng SQL Injection:

- Dữ liệu được nhập vào (input) từ người dùng phải được truyền dưới dạng tham số, không được sử dụng cách cộng xâu trong các truy vấn tới cơ sở dữ liệu.

2.3 Xử lý dữ liệu đầu vào tránh lỗ hổng XSS:

- Encode dưới dạng HTML các ký tự đặc biệt do người dùng gửi lên máy chủ và các ký tự đặc biệt trong cơ sở dữ liệu trước khi gửi tới người dùng.

2.4 Sử dụng Token trong các phương thức GET và POST tránh lỗ hổng CSRF:

- Trong chức năng mà người dùng tương tác với cơ sở dữ liệu thông qua các form, liên kết. Phải sử dụng thêm biến token (được tạo ra mỗi đầu phiên truy cập của người dùng) như một tham số trong phương thức GET hoặc POST và kiểm tra giá trị token này tại máy chủ để xác nhận hành vi của người dùng.

2.5 Kiểm soát các thao tác với file:

- Giới hạn chỉ cho phép các định dạng file theo yêu cầu của ứng dụng được phép upload lên máy chủ. Kiểm soát file upload ở phía máy chủ. Lưu trữ các file upload tại một thư mục riêng nằm ngoài thư mục web hoặc không cho phép truy cập, thực thi trên các thư mục đó.
- Chặn các kí tự \, /, null và kiểm tra phần mở rộng của file khi xử lý với tên file trên máy chủ.

2.6 Mã hóa dữ liệu nhạy cảm

- Những dữ liệu nhạy cảm trong cơ sở dữ liệu cần phải được mã hóa.

2.7 Kiểm tra quyền truy cập của người dùng

- Kiểm soát quyền của người dùng trong mỗi request lên máy chủ.

2.8 User enumeration

- Sử dụng chung thông báo lỗi cho cả 2 trường hợp nhập sai tên đăng nhập và mật khẩu trên trang đăng nhập vào hệ thống. Nhằm tránh trường hợp thông báo lỗi trên trang đăng nhập phân biệt giữa nhập sai tên đăng nhập và sai mật khẩu → Dựa vào đó kẻ tấn công có thể thử và tìm ra các user có trên hệ thống.
- Sử dụng captcha cho các chức năng đăng ký, reset/forgot mật khẩu để tránh các công cụ tự động khai thác lỗi user enumeration.

2.9Session fixation

- Tạo mới lại phiên người dùng (renew session) sau khi đăng nhập và xóa session cũ (trên Server) sau khi log out. Tránh trường hợp Session của ứng dụng trước và sau khi đăng nhập/đăng xuất không thay đổi

2.10 Sử dụng cookie an toàn

- Yêu cầu thiết lập thuộc tính "HTTPOnly" cho session cookie. Vì nếu Session cookie không được set thuộc tính "HTTP Only". Kẻ tấn công có thể sử dụng mã javascript để đánh cắp cookie của người dùng.
- Với các website sử dụng HTTPS cho các chức năng quan trọng, HTTP cho các chức năng thông thường, thì cần thiết lập thuộc tính "secure" cho các session cookie để đảm bảo session cookie này được truyền qua kênh truyền có mã hóa. Nếu không sử dụng thuộc tính này thì dù có sử dụng HTTPS cho chức năng quan trọng như đăng nhập, nhưng phiên làm việc vẫn có thể bị nghe lén, lấy cắp khi truyền qua HTTP.

2.11 Chuyển hướng và chuyển tiếp thiếu thẩm tra

- Không cho phép người dùng cuối có thể can thiệp vào quá trình redirect từ ứng dụng web này sang ứng dụng web khác. Nếu cần sử dụng thì URL phải được kiểm tra, đảm bảo URL được redirect đến nằm trong danh sách cho phép của ứng dụng.

2.12 Để lộ dữ liệu của hệ thống

- Yêu cầu tất cả dữ liệu, tài nguyên hệ thống (báo cáo, file lưu trữ khi xuất hệ thống xuất ra hoặc tải lên, file cấu hình...) không được lưu trong thư mục cho phép truy cập trực tiếp không qua xác thực. Việc thực hiện tải (download) dữ liệu này phải qua bước xác thực và tham số trong link download phải được mã hóa, tránh để lộ đường dẫn tuyệt đối của file dữ liệu trên hệ thống.
- Đối với các hệ thống đang triển khai yêu cầu các dữ liệu, tài nguyên hệ thống

(báo cáo, file lưu trữ khi xuất hệ thống xuất ra hoặc tải lên...) phải được lưu trong thư mục bên ngoài thư mục chứa source code của web server.

2.13 Thất thoát thông tin do kiểm soát lỗi và ngoại lệ không tốt

- Khi xử lý ngoại lệ và thông báo miêu tả lỗi yêu cầu chương trình không hiển thị chi tiết lỗi cho người dùng cuối, hạn chế thông tin hiển thị nhất có thể. Việc này nhằm ngăn cản kẻ tấn công dựa vào các thông báo lỗi chương trình để thu thập thông tin về hệ thống. Các thông tin lỗi này phải được log lại bên server để phục vụ bảo trì, xác định nguyên nhân lỗi ứng dụng...

2.14 Sử dụng Captcha an toàn

- Với các chức năng quan trọng, ảnh hưởng đến dữ liệu quan trọng cần tránh chương trình tự động có thể thực thi các chức năng này thì cần sử dụng captcha để xác thực hành động thực thi là do người sử dụng thực hiện. Ví dụ như các chức năng: Đổi mật khẩu (renew password), cấp lại mật khẩu (reset password), nhớ lại mật khẩu (forgot password), chức năng đăng ký người dùng, chức năng trừ, cộng tiền, chức năng xóa hủy tài khoản, ...

2.15 File inclusion

- Kiểm tra chặt chẽ các hàm include, require, require_once, include_once cho phép gọi các file khác trong hệ thống.

2.16 Command injection

- Hạn chế tối đa sử dụng dữ liệu do người dùng đưa vào để gửi đến thực thi các lệnh của hệ điều hành.
- Trường hợp bắt buộc phải sử dụng cơ chế trên thì cần validate chặt chẽ dữ liệu người dùng gửi vào, tránh các ký tự cho phép nối thêm các câu lệnh thực thi của hệ điều hành. Sử dụng whitelist là danh sách chứa các ký tự được phép xuất hiện trong dữ liệu, để so sánh, đối chiếu.

2.17 Xml/Xpath injection

- Escape hoặc encode XML dữ liệu trước khi đưa vào file dữ liệu dạng XML, để tránh hacker có thể chèn dữ liệu có chứa thẻ xml gây ra sai lệch dữ liệu trong file xml.
- Dữ liệu input từ người dùng phải được tham số hóa trước khi đưa vào thư viện xử lý Xpath.

2.18 Các lỗi liên quan đến xử lý luồng nghiệp vụ, logic.

- Lập trình viên khi xây dựng, phát triển ứng dụng cần lắm rõ luồng nghiệp vụ, logic của ứng dụng, xử lý các case mà kẻ tấn công có thể lợi dụng thực hiện viết

chương trình tự động để thực hiện.

PHÒNG CÔNG NGHỆ THÔNG TIN	P. TGĐ CHUYÊN TRÁCH
Thiếu tá Trần Anh Quân	Đại tá Tống Viết Trung

Nơi *nhận:*
- *Ban TGĐ;*
- *Các Cty, TT;*
- *Lưu CNTT, Toàn 02.*

PHỤ LỤC: CÁCH THỰC HIỆN

1. Cập nhật thư viện (Lib) bản vá cho web framwork

- **Nguy cơ:** Các ứng dụng web thường được phát triển bằng các framwork (struts, adempiere, zk...) các thư viện (Lib) sử dụng trong các framwork version cũ có nhiều các lỗi về ATTT đã được public, kẻ tấn công có thể lợi dụng để khai thác tấn công hệ thống, như lỗi (Struts2 Remote Command Execution...)
- **Khắc phục:** Thực hiện cập nhật các thư viện mới nhất, cập nhật bản vá ATTT cho các thư viện, framework sử dụng.

2. Kiểm soát truy vấn cơ sở dữ liệu để tránh lỗ hổng SQL Injection

- **Nguy cơ:** Khi truy vấn tới cơ sử dữ liệu lập trình viên thường sử dụng cách cộng xâu Input từ người dùng, các câu truy vấn này có thể bị mắc lỗi SQL Injection hoặc HQL Injection (nếu sử dụng Hibernate). Bằng việc lợi dụng các lỗi này, kẻ tấn công có thể xem, thêm, sửa, xóa dữ liệu trong database từ đó chiếm được tài khoản admin, lấy cắp thông tin người dùng...
- **Phòng chống:**
  - Truy vấn SQL phải dùng PrepareStatement, tất cả tham số phải được add bằng hàm (setParam..), không được xử dụng cách cộng xâu trong truy vấn.
  - Truy vấn SQL tất cả tham số phải được add bằng hàm (setParam), không được xử dụng cách cộng xâu trong truy vấn.
  - Với một số trường hợp sử dụng ORDER BY không thể dùng được hàm setParam thì có thể định nghĩa một mảng chứa toàn bộ các column (field) cần ORDER BY gọi là whitelist. Mỗi khi cần ORDER BY thì kiểm tra lại xem column (field) đó có thuộc mảng whitelist đã định nghĩa không.

Ví dụ 1: Đoạn code kiểm tra đăng nhập với username/password do người dùng nhập vào

```
String sql = "select * from users where user_name = '" + userName
        + "' and password = '" + encrypt(password) + "'";
Statement statement = connection.createStatement();
ResultSet rs = statement.executeQuery(sql);
if (!rs.next()) {
      bResult = false;
} else {
      bResult = true;
}
```

Nhập vào username là *test' or '1'='1* thì câu query sẽ là: *select * from users where user_name='test' or '1'='1' and password='...'*. Mệnh đề where sẽ tương đương với *user_name = 'test'*. Như vậy dù không có password vẫn đăng nhập được vào hệ thống. Đoạn code bên dưới, username, password được tham số hóa khi đưa vào câu truy vấn nên tránh được lỗi SQL Injection:

```
String sql = "select * from users where user_name = ? and password = ?";
PreparedStatement statement = connection.prepareStatement(sql);
statement.setString(0, userName);
statement.setString(1, encrypt(password));
ResultSet rs = statement.executeQuery(sql);
if (!rs.next()) {
     bResult = false;
} else {
     bResult = true;
}
```

Ví dụ 2: Một số trường hợp không thể ngăn chặn được lỗi SQL injection qua lệnh "order by". Do không sử dụng được hàm setParam có thể sử dụng phương pháp sau:

```
// Mảng lưu danh sách các column (field) của BO cần order by (hay gọi là
whitelist)
private static List columnSort = new ArrayList();
public static String getColumnSort(String sortField) {
               // Thực hiện 1 lần và lấy ra toàn bộ mảng column cần order
và  add vào whitelist
      if (columnSort.size() == 0) {
      // Danh sách BO cho phép order by
      String[] arrTableName = {"ActionLog",
      "BanPosition",
      "Category",
      ........
      };
// Lấy ra toàn bộ các column (field) BO cần order by
      for (String tableName : arrTableName) {
            try {
                  Class                        cls                        =
Class.forName("com.demo.DEMOATTT.database.BO." + tableName);
                  Field[] fieldArr = cls.getDeclaredFields();
                        for (int i = 0; i < fieldArr.length; i++) {
                        String fieldName = fieldArr[i].getName();
                        // add các column vào 1 mang
                        columnSort.add(fieldName);
                        }
            } catch (ClassNotFoundException ex) {
```

```
                }
        }
        }
        // Cắt ký tự "-" ở đầu field sort
        String sort = sortField;
        if (sortField != null && sortField.startsWith("-")) {
        sortField = sortField.substring(1);
        }
        // Kiểm tra field cần order by có nằm trong danh sách field cho
phép sort hay không
                if (sortField != null && columnSort.contains(sortField)) {
        return sort;
        }
        return null;
}
```

Ví dụ 3: Đoạn code .Net có lỗi SQL injection sử dụng cộng xâu GenreId, và SongName vào câu truy vấn:

```
String connectionString =
System.Configuration.ConfigurationSettings.AppSetting["connectionString"];
sqlConnection connection = new SqlConnection(connectionString);

//ERROR
String query = "select top 10 * from dbo.MO_Items where GenreID=" +
paramIdGetFromQuery + "and SongName like '%' + paramSongNameGetFromQuery +
'%';

//PROCESS
SqlCommand selectCommand = new SqlCommand(query,connection);
DataSet dataset = new DataSet();
SqlDataAdapter adapter = new SqlDataAdapter();
adapter.SelectCommand = selectCommand;
adapter.Fill(dataset);
```

Đoạn code an toàn sau khi sửa lỗi, bằng cách add parameter cho 2 biến GenreId, và SongName cần truyền vào câu truy vấn:

```
String connectionString =
System.Configuration.ConfigurationSettings.AppSetting["connectionString"];
sqlConnection connection = new SqlConnection(connectionString);

//FIX ERROR SQL INJECTION
String query = "select top 10 * from dbo.MO_Items where GenreID=@GenreID
and SongName like @song";
SqlCommand = new SqlCommand(query, connection);
```

```
//Dinh nghia cac loai tham so la int hay string
selectCommand.Parameters.Add("GenreID", SqlDbType.Int);
selectCommand.Parameters.Add("@song", SqlDbType.NVarChar);
//Gan gia tri cac tham so
selectCommand.Parameters["@GenreID"].Value =
Convert.ToInt32(paramIdGetFromQuery);
selectCommand.Parameters["@song"].Value = "%" + paramSongNameGetFromQuery
+ "%";// hoac like '%' + @song + '%'

//PROCESS
DataSet dataset = new DataSet();
SqlDataAdapter adapter = new SqlDataAdapter();
adapter.SelectCommand = selectCommand;
adapter.Fill(dataset);
```

Ví dụ 4: Đoạn code PHP có lỗi SQL injection sử dụng cộng xâu username, password vào câu truy vấn:

```
// Connect
$link = mysql_connect('mysql_host', 'mysql_user', 'mysql_password')
      OR die(mysql_error());

// Query database to check if there are any matching users
$query = "SELECT * FROM users WHERE user='{$_POST['username']}'" .
            " AND password='{$_POST['password']}'";
mysql_query($query);
```

Đoạn code an toàn sau khi sửa lỗi bằng cách sử dụng prepare để bind parameter 2 tham số user, password:

```
// Connect
$link = mysql_connect('mysql_host', 'mysql_user', 'mysql_password')
      OR die(mysql_error());

// Query
$query = $link->prepare("SELECT * FROM users WHERE user=:user AND
password=:password");
$query->bindParam(':user',$_POST['username']);
$query->bindParam(':password',$_POST['password']);
$query->execute();
```

3. Xử lý dữ liệu đầu vào để tránh lỗ hổng XSS

- **Nguy cơ:** Kết quả server trả về cho người dùng chủ yếu dưới dạng HTML. Nội dung trả về thường bao gồm cả những giá trị mà người dùng nhập vào hệ thống có

thể bị mắc lỗi XSS nếu không kiểm soát dữ liệu đầu vào. XSS (Cross-Site Scripting) là một kĩ thuật tấn công bằng cách chèn vào các website động (ASP, PHP, CGI, JSP ...) những thẻ HTML hay những đoạn mã script nguy hiểm có thể gây nguy hại cho những người sử dụng khác. Trong đó, những đoạn mã nguy hiểm được chèn vào hầu hết được viết bằng các Client-Site Script như JavaScript, JScript, DHTML và cũng có thể là cả các thẻ HTML.

- **Phòng chống:**

- Encode dưới dạng HTML các ký tự đặc biệt do client gửi đến bao gồm: <,>,&,’,”,/ trong các trường hợp
  - Dữ liệu client gửi lên máy chủ
  - Dữ liệu lấy ra từ database khi trả về cho client

  Bản chất của việc encode là thay thế các kí tự trên bằng chuỗi tương ứng trong bảng bên dưới:

STT	Ký tự	HTML
1	"	&quot;
2	&	&amp;
3	'	&#x27;
4	/	&#x2F;
5	<	&lt;
6	>	&gt;

Ví dụ 1: Trang jsp bên dưới hiển thị lên lời chào với tên người dùng được lấy từ client

```
<html>
    <head>
        <meta   http-equiv="Content-Type"   content="text/html;charset=UTF-
8">
        <title>JSP Page</title>
    </head>
    <body>
        <%
            String user = request.getParameter("user");
            request.setAttribute("user", user);
        %>
      <h1>Hello ${user} !</h1>
    </body>
</html>
```

Khi nhập vào địa chỉ trình duyệt *http://localhost/example?user=abc* thì trên trình duyệt sẽ hiện thì dòng *Hello abc !*

Khi nhập vào địa chỉ trình duyệt

http://localhost/example?user=abc</h1><script>alert('XSS')</script> thì trên trên trình duyệt sẽ thực hiện đoạn java script thông báo XSS.

Để khắc phục lỗi này ta có thể dùng thư viện JSTL để encode HTML biến user

```
<%@taglib prefix="fn" uri="http://java.sun.com/jsp/jstl/functions"%>
<html>
    <head>
        <meta   http-equiv="Content-Type"   content="text/html;charset=UTF-
8">
        <title>JSP Page</title>
    </head>
    <body>
        <%
            String user = request.getParameter("user");
            request.setAttribute("user", user);
        %>
      <h1>Hello ${fn:escapeXml(user)} !</h1>
    </body>
</html>
```

Ví dụ 2: Code .Net không encode dữ liệu đầu ra *message* trong file .aspx, có nguy cơ bị mắc lỗi XSS:

```
/* File .aspx */
  <div>
    Result: <%=message%>
  </div>
  <div>
     <asp:Label runat="server" ID="lbResult"></asp:Label>
  </div>
```

Để khắc phục lỗi trên, có thể thực hiện encode đầu ra trong file .aspx hoặc file .aspx.cs như sau:

- Thực hiện encode đầu ra file .aspx

```
<div>
    Result: <%=Microsoft.Security.Application.AntiXss.HtmlAttributeEncode(message)%>
</div>
```

- Thực hiện encode đầu ra file .aspx.cs

```
//Thẻ Label ID=lbResult được xử lý trước khi bind ra dữ liệu.
lbResult.Text = Microsoft.Security.Application.AntiXss.HtmlAttributeEncode(message);
```

Ví dụ 3: Đoạn code PHP hiển thị nội dung biến *content* không được encode, gây ra lỗi XSS:

```
//XSS PHP File
$content = $_REQUEST['content'];
echo $content;
```

Khắc phục lỗi bằng cách encode biến *content* trước khi hiển thị bằng hàm htmlspecialchars:

```
//XSS PHP File
$content = $_REQUEST['content'];
echo htmlspecialchars($content);
```

4. Sử dụng token để tránh lỗ hổng CSRF

- **Nguy cơ:** CSRF (Cross-site request forgery) là phương pháp mượn quyền của người dùng khác để thực hiện một hành động không cho phép. Ví dụ: Để có thể xóa một bài viết trên diễn đàn một member có thể mượn tay của một admin để làm việc đó vì member không đủ chủ quyền nhưng admin lại đủ chủ quyền để thực hiện hành động này. Kẻ tấn công lừa admin truy cập vào trang web có chứa đoạn mã xóa bài viết trên diễn đàn (Admin đang đăng nhập vào diễn đàn) như vậy admin đã gửi yêu cầu xóa bài viết trên diễn đàn mà không hề biết.
- **Phòng chống**: Đối với các yêu cầu ảnh hưởng tới dữ liệu, quá trình hoạt động và có khả năng làm mất an toàn thông tin của hệ thống. VD: yêu cầu đọc, ghi, sửa, xóa thông tin, dữ liệu hệ thống phải sử dụng thêm biến token. Trên server sẽ kiểm tra token trong yêu cầu gửi lên từ client, nếu token không hợp lệ thì yêu cầu sẽ không được thực hiện.

Ví dụ 1: Ứng dụng struts cho phép hiển thị lời chào với tên người dùng nhập từ form index.jsp.

```
<%@taglib prefix="s" uri="/struts-tags" %><br />
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-
8">
    </head>
    <body>
        <h2>Hello World!</h2>
        Struts 2 Message: <s:property value="message" default="Guest." />
        <s:form method="GET" action="/HelloStruts2World.action">
            Enter your name:<s:textfield name="userName" />
            <s:submit value="Submit" />
        </s:form>
    </body>
</html>
```

struts.xml

```
<struts>
    <package name="/" extends="struts-default">
        <action name="HelloStruts2World" class="hello.HelloStruts2World">
            <result name="success">/index.jsp</result>
        </action>
    </package>
</struts>
```

helloStruts2World.java

```
package hello;
import com.opensymphony.xwork2.ActionSupport;
public class HelloStruts2World extends ActionSupport {
    private String userName;
    public String getUserName() {
        return userName;
    }
    public void setUserName(String userName) {
        this.userName = userName;
    }
    private String message;
    public String getMessage() {
        return message;
    }
    @Override
    public String execute() {
        message = "Hello, " + userName + ".";
        return SUCCESS;
    }
}
```

Tuy nhiên, yêu cầu trên có thể được thực hiện mà không cần phải nhập username từ form bằng cách đưa trực tiếp vào URL:

http://localhost:8084/TestStruts/HelloStruts2World.action?userName=abc

Để khắc phục lỗi trên, ta có thể sử dụng token intercepter đã có sẵn của struts.

index.jsp

```
<%@taglib prefix="s" uri="/struts-tags" %><br />
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-
8">
    </head>
    <body>
        <h2>Hello World!</h2>
```

```
        Struts 2 Message: <s:property value="message" default="Guest." />
        <s:form method="GET" action="/HelloStruts2World.action">
            <s:token/>
            Enter your name:<s:textfield name="userName" />
            <s:submit value="Submit" />
        </s:form>
    </body>
</html>
```

struts.xml

```
<struts>
    <package name="/" extends="struts-default">
        <interceptors>
            <interceptor-stack name="defaultSecurityStack">
                <interceptor-ref name="defaultStack" />
                <interceptor-ref name="tokenSession">
                    <param name="excludeMethods">*</param>
                </interceptor-ref>
            </interceptor-stack>
        </interceptors>
        <default-interceptor-ref name="defaultSecurityStack" />
        <global-results>
            <result name="invalid.token">/error.jsp</result>
        </global-results>
        <action name="HelloStruts2World" class="hello.HelloStruts2World">
            <interceptor-ref name="defaultSecurityStack">
                <param name="tokenSession.includeMethods">*</param>
            </interceptor-ref>
            <result name="success">/index.jsp</result>
        </action>
    </package>
</struts>
```

Ví dụ 2: Ứng dụng .Net mắc lỗi CSRF khi cho phép người dùng nhập 1 số tiền vào ô textbox, rồi thực hiện gửi tiền bằng cách click vào nút SendMoney như sau:

File SendMoney.ASPX

```
<form id="from1" runat="server">
<div>
      <asp:TextBox ID="textMoney" runat="server" />
      <asp:Button ID="btnSendMoney" runat="server" Text="SendMoney"
            onclick="btnSendMoney_Click" />
</div>
</form>
```

File SendMoney.Aspx.cs

```
protected void btnSendMoney_Click(object sender, EventArgs e)
{
    Response.Write("Đã gửi số tiền là: "+txtMoney.Text.Trim());
}
```

Ứng dụng trên mắc lỗi CSRF, kẻ tấn công có thể tự tạo request hàng loạt gửi lên server với tham số là số tiền cần gửi mà không cần phải nhập trên ô textbox trên form gửi tiền.

Để khắc phục lỗi CSRF này, server cần sinh ra token gửi về web browser, và yêu cầu gửi lên token mỗi khi thực hiện submit để kiểm tra xem có đúng là token mà server đã gửi về hay không. Đoạn code sửa lỗi như sau:

File SendMoney.aspx

```
<form id="from1" runat="server">
<div>
      <asp:HiddenField runat="server" ID="hiddenToken" />
<asp:TextBox ID="textMoney" runat="server" />
<asp:Button ID="btnSendMoney" runat="server" Text="SendMoney"
onclick="btnSendMoney_Click" />
</div>
</form>
```

File SendMoney.aspx.cs

```
private void ResetToken()
{
      session["token"] = Guid.NewGuid().ToString();
      hiddenToken.Value = Session["token"].Tostring();
}

Private Boolean CheckPermissionByToken()
{
      if(session["token"].ToString()==hiddenToken.Value)
      {
            ResetToken();//sau khi check la thuc hien reset luon
            return true;
      }
      return false;
```

```
}

protected void Page_Load(object sender, EventArgs e)
{
      if(Page.IsPostBack==false)
      {
            //lan dau tien vao trang khoi tao token
            ResetToken();
      }
}

Protected void btnSendMoney_Click(object sender, EventArgs e)
{
      if(CheckPermissionByToken())//kiemtra xem co quyen hop le khong
      {
            Response.Write("Đã gửi số tiền là: " + txtMoney.Text.Trim());
      }
      else
      {
            Response.Write("Bạn vừa bị tấn công CSRF");
      }
}
```

Ví dụ 3: Ứng dụng PHP mắc lỗi CSRF khi cho phép người dùng mua cổ phiếu bằng cách nhập mã cổ phiếu và số lượng gửi lên server như sau:

Form html:

```
<form action="buy.php" method="POST">
     <p>Symbol: <input type="text" name="symbol" /></p>
     <p>Shares: <input type="text" name="shares" /></p>
     <p><input type="submit" value="Buy" /></p>
</form>
```

File buy.php:

```
<?php
session_start();
if (isset($_REQUEST['symbol'] && isset($_REQUEST['shares']))
{
      buy_stocks($_REQUEST['symbol'],
      $_REQUEST['shares']);
}
 ?>
```

Ứng dụng mắc lỗi CSRF, cho phép hacker có thể tạo ra hàng loạt request mua cổ phiếu trực tiếp lên server dạng: `http://example.org/buy.php?symbol=SCOX&shares=1000` mà không cần nhập từ form mua trên trình duyệt.

Để khắc phục lỗi CSRF này, cần bổ sung token như sau:

Form html:

```
<?php
      $token = md5(uniqid(rand(), TRUE));
      $_SESSION['token'] = $token;
      $_SESSION['token_time'] = time();
?>
<form action="buy.php" method="post">
      <input type="hidden" name="token" value="<?php echo $token; ?>" />
      <p>
      Symbol: <input type="text" name="symbol" /><br />
      Shares: <input type="text" name="shares" /><br />
      <input type="submit" value="Buy" />
      </p>
</form>
```

File buy.php:

```
<?php
$token_age = time() - $_SESSION['token_time'];
if ($token_age <= 300)
{
/* Less than five minutes has passed. */
      if ($_POST['token'] == $_SESSION['token'])
      {
      /* Valid Token */
            if (isset($_REQUEST['symbol'] && isset($_REQUEST['shares']))
            {
                  buy_stocks($_REQUEST['symbol'],
                  $_REQUEST['shares']);
            }
      }
}
?>
```

5. *Kiểm soát file upload lên hệ thống*

- **Nguy cơ:** Các thao tác với file thường sử dụng tên file, đường dẫn file được gửi lên từ client, nếu ứng dụng không kiểm soát tốt các giá trị này (việc kiểm soát phải được thực hiện phía server) có thể dẫn đến việc download hoặc upload các file không hợp lệ.
- **Phòng chống:** Kiểm soát phía server tên file, đường dẫn file được gửi lên từ client.

- Kiểm soát phần mở rộng của file, chỉ cho phép thực hiện với các file có định dạng theo yêu cầu. Không bắt buộc phải kiểm tra nội dung file.
- Các hàm liên quan đọc ghi file, biến đường dẫn file phải được lọc /, \ và kí tự null.
- Phần filename ban đầu người dùng upload lên server phải bỏ đi, dùng 1 chuỗi mới ngẫu nhiên thay thế cho tên file. Tên này được sinh ra ngẫu nhiên không được dùng các thuật toán md5, sha256... Thay vào đó có thể sử dụng các hàm sinh chuỗi ngẫu nhiên có sẵn trong ngôn ngữ lập trình để sinh ra tên.

Ví dụ 1: Tạo tên file cho file ảnh định dạng jpg với hàm UUID có sẵn như sau.

```
#!/usr/bin/env python
import uuid
filename='%032x' % uuid.uuid4() + ".jpg"
```

Tên file ban đầu là shell.php.jpg upload lên server thì đổi thành

6817c84abd3d4c9abfee21a01cb39b98.jpg

Ví dụ 2: Đoạn code bên dưới in ra đường dẫn file với đầu vào là tên file

```
String fileName = "temp.txt";
File file1 = new File(fileName);
System.out.println("File 1 path: " + file1.getCanonicalPath());
fileName = "./../../../../../../../boot.ini";
File file2 = new File(fileName);
System.out.println("File 2 path: " + file2.getCanonicalPath());
fileName = "boot.ini" + String.valueOf((char) 0) + ".txt";
File file3 = new File(fileName);
System.out.println("File 3 path: " + file3.getCanonicalPath());
```

Với đường dẫn thư mục hiện tại là

*"*C:\Documents and Settings\Website\upload\test*"*.Ta có kết quả thực hiện:

```
File 1 path: C:\Documents and Settings\Website\upload\test\temp.txt
File 2 path: C:\boot.ini
File 3 path: C:\Documents and Settings\Website\upload\test\boot.ini
```

Trong trường hợp 1, kết quả in ra đường dẫn file *temp.txt* nằm trong thư mục hiện tại

Trong 2 trường hợp còn lại, tên file được thay đổi để truy cập đến các file không được phép.

- Trường hợp 2: Trong tên file có chứa các chuỗi ./ và ../, các chuỗi này có tác dụng chuyển đến thư mục hiện tại (./) và thư mục cha của thư mục hiện tại (../). Vì thế

với tên file là "*./../../boot.ini*", kết quả in ra là file *boot.ini* nằm trong thư mục gốc ổ C. ***Chú ý:*** Kỹ thuật này thường được dùng để truy cập đến các thư mục nằm ngoài thư mục hiện tại. Các chuỗi .\ và ..\ cũng có tác dụng tương tự như *./* và *../*

- Trường hợp 3: Các hàm xử lý file của hệ điều hành khi gặp kí tự NULL (mã ASCII là 0) trong tên file sẽ hiểu rằng đây là kí tự kết thúc xâu chứa tên file và bỏ qua tất cả các kí tự phía sau (đặc điểm của các hàm xử lý xâu bằng ngôn ngữ C – ngôn ngữ của hầu hết các hệ điều hành). Vì thế kết quả trong trường hợp này sẽ là file *boot.ini* trong thư mục hiện tại mặc dù tên file nhập vào có phần mở rộng là "*.txt*".

  Chú ý: Kỹ thuật này thường được dùng để vượt qua việc chặn phần mở rộng của file.

Để sửa lỗi trên ta có thể sử dụng hàm lọc các kí tự /,\,null trong tên file.

```
public static String getSafeFileName(String input) {
        StringBuilder sb = new StringBuilder();
        for (int i = 0; i < input.length(); i++) {
            char c = input.charAt(i);
            if (c != '/' && c != '\\' && c != 0) {
                sb.append(c);
            }
        }
        return sb.toString();
    }
```

Ví dụ 3: Đoạn mã bên dưới có tác dụng upload file từ client và save vào thư mục upload.

```
private File clientFile;
private String clientFileFileName;
@Override
public String execute() throws Exception {
    try {
        if (clientFileFileName != null) {
            clientFileFileName = getSafeFileName(clientFileFileName);
            if (!clientFileFileName.endsWith(".txt")) {
                message = "Not .txt file!";
            } else {
                String uploadDir = "C:\\upload\\";
                File uploadFile = new File(uploadDir +
clientFileFileName);
```

```
                FileUtils.copyFile(clientFile, uploadFile);
                message = "Upload file success!";
            }
        }
    } catch (Exception e) {
        message = "Upload file error!";
    }
    return SUCCESS;
}
```

Sử dụng hàm getSafeFileName trong ví dụ 1 kết hợp với việc kiểm tra phần mở rộng đảm bảo upload file an toàn, chỉ các file được phép mới được upload lên server và các file này chỉ có thể được phép save vào thư mục chỉ định.

Ví dụ 4: Hàm getSafeFileName trên .Net:

```
  private static String getSafeFileName(String input)
   {
        StringBuilder sb = new StringBuilder();
        for(int i=0; i < input.Length; i++)
        {
            char c = input[i];
            if(c!='/' && c!='\\' && c!=0)
            {
                sb.Append(c);
            }
        }
        return sb.ToString();
   }
```

Ví dụ 5: Hàm getSafeFileName trên PHP:

```
function getSafeFileName($input){
      $result = '';
      for ($i=0; $i<strlen($input); $i++){
            $c = substr($input, $i, 1);
            if($c !='/' && $c != '\\' && $c !='\0' && $c != null){
                $result = $c
            }
      }
}
```

6. Mã hóa dữ liệu nhạy cảm

- **Nguy cơ:** Khi hệ thống bị tấn công và kẻ tấn công lấy được thông tin trong cơ sở dữ liệu, các dữ liệu nhạy cảm sẽ bị lộ nếu không được mã hóa hoặc mã hóa không an toàn.

  Note: Các thông tin sau được cho là nhạy cảm: Thông tin về tài khoản/mật khẩu, thẻ ngân hàng, thông tin về tiền…

- **Phòng chống:**

  - Mã hóa các dự liệu nhạy cảm trong cơ sở dữ liệu.
  - Các hàm mã hóa 1 chiều phải có thêm *salt.* Chú ý*: salt* phải đảm bảo tính ngẫu nhiên

Ví dụ: Mật khẩu của người dùng được mã hóa trong cơ sở dữ liệu bằng hàm mã hóa 1 chiều. Kẻ tấn công lấy được mật khẩu đã mã hóa là *QL0AFWMIX8NRZTKeof9cXsvbvu8=* và có thể tìm được mật khẩu chưa mã hóa là *123*, thuật toán mã hóa là *hash = base64(SHA1(pass))* bằng cách tìm kiếm trên Internet. Các mật khẩu phổ biến dễ bị dò ngược lại bằng phương pháp này.

Để sửa lỗi này ta cần thêm *salt* vào hàm mã hóa. Thay vì *hash = encrypt(pass)*, hàm mã hóa sẽ chuyển thành *hash = encrypt(salt + pass)* và khi bị lộ hash cũng không thể tìm kiếm được trên Internet để tìm ra mật khẩu chưa mã hóa. *Salt* được sinh ngẫu nhiên cho từng user, và được lưu trong CSDL.

7. Kiểm tra quyền truy cập của người dùng

- **Nguy cơ:** Trong các hệ thống có phân quyền, mỗi người dùng chỉ được phép truy cập các chức năng, các dữ liệu mà mình được phép. Tuy nhiên, nếu việc kiểm tra quyền không được kiểm soát tốt thì người dùng có thể truy cập được các chức năng, các dữ liệu không được quyền.
- **Phòng chống:** Kiểm tra quyền trong từng request gửi lên server. Việc kiểm tra quyền gồm 2 nội dung kiểm tra là :
  - Người dùng có được phép thực thi chức năng theo request hay không?
  - Nếu người dùng được phép thực thi chức năng thì kiểm tra tiếp người dùng có được phép thao tác chức năng đó trên dữ liệu trong request hay không?

Ví dụ: Hàm bên dưới mắc lỗi không kiểm tra quyền của người dùng, mà cho phép luôn quản trị khóa tài khoản người dùng với đầu vào là ID của người dùng được gửi từ client.

```
public String deleteFileofUser() {
```

```
        String strFileId = getRequest().getParameter("fileId");
        Long fileId = Long.parseLong(strFileId);
        doDeleteFileofUser(fileId);
        return SUCCESS;
    }
```

Như vậy, bằng cách thay đổi ID của người dùng từ client, người quản trị này có thể khóa tài khoản của các người dùng mà mình không được phép quản lý.

Để sửa lỗi này, ta cần kiểm tra quyền khóa của quản trị đối với người dùng này trước khi thực hiện khóa.

```
public String deleteFileofUser() {
        String strFileId = getRequest().getParameter("fileId");
        Long fileId = Long.parseLong(strFileId);
        if (checkLockPermission(userId)) {
              doDeleteFileofUser(fileId);
              return SUCCESS;
        } else {
            return ERROR;
        }

    }
```

8. User enumeration

- **Nguy cơ:**
  - Trường hợp thông báo lỗi trên trang đăng nhập phân biệt giữa nhập sai tên đăng nhập và sai mật khẩu -> Dựa vào đó hacker có thể thử và tìm ra các user có trên hệ thống.
  - Với các chức năng phải thông báo tên user nhập vào là đúng hay sai như các chức năng reset password, forgot password, chức năng đăng ký thì hacker có thể thử và tìm ra các user có trên hệ thống.
- **Phòng chống:**
  - Sử dụng chung thông báo lỗi cho cả 2 trường hợp nhập sai tên đăng nhập và mật khẩu trên trang đăng nhập vào hệ thống.
  - Sử dụng captcha cho các chức năng đăng ký, reset, forgot mật khẩu để tránh các công cụ tự động khai thác lỗi user enumeration.

Ví dụ: Ứng dụng mắc lỗi user enumeration do thông báo user không tồn tại, hacker có thể dựa vào đó để dò đoán tên user:

```
private String checkLogin(String user, String password) {
    User userDB = null;
    if (user != null && !"".equals(user)) {
        userDB = getUser(user);
    }
    if (userDB == null) {
        return "Không tồn tại user";
    }
    if (userDB.getPassword()!= password) {
        return "Sai mật khẩu";
    }
    return "Success";
}
```

Khắc phục lỗi trên bằng cách đưa ra thông báo chung cho cả 2 trường hợp sai mật khẩu và user, làm hacker không đoán được tên user như sau:

```
private String checkLogin(String user, String password) {
        User userDB = null;
        if (user != null && !"".equals(user)) {
            userDB = getUser(user);
        }
        if (userDB == null) {
            return "Sai tên đang nhập hoặc mật khẩu";
        }
        if (userDB.getPassword()!= password) {
            return " Sai tên đang nhập hoặc mật khẩu ";
        }
        return "Success";
    }
```

9. Sesion fixation

- **Nguy cơ:** Kỹ thuật tấn công cho phép hacker mạo danh người dùng hợp lệ bằng cách gửi một sesion ID hợp lệ đến người dùng, sau khi người dùng đăng nhập vào hệ thống thành công, hacker sẽ dùng lại sesion ID đó và nghiễm nhiêm trở thành người dùng hợp lệ.
- **Phòng chống:**
  - **Biện pháp 1**: Chống việc đăng nhập với một session ID có sẵn:
    + Theo kiểu tấn công này, người dùng đăng nhập vào hệ thống thông qua một session ID do hacker tạo sẵn thay vì cho trình chủ tạo mới, do đó để có phòng chống, ứng dụng phải hủy bỏ session ID được cung cấp bởi trình duyệt của

người dùng khi đăng nhập và luôn tạo một session ID mới khi người dùng đăng nhập thành công.

- **Biện pháp 2:** Giới hạn phạm vi ứng dụng của session ID.
  + Kết hợp session ID với địa chỉ của trình duyệt. Chú ý trường hợp mạng client có sử dụng NAT để truy cập vào server ứng dụng sẽ không dùng được phương pháp này.
  + Xóa bỏ session khi người dùng thoát khỏi hệ thống hay hết hiệu lực, có thể thực hiện trên trình máy chủ.
  + Thiết lập thời gian hết hiệu lực cho session, tránh trường hợp hacker có thể duy trì session và sử dụng nó lâu dài.

Ví dụ: Khi user đăng nhập vào thành công, sửa lại cookie cho user như sau:

Java:

```
private String checkLogin(String user, String password) {
        User userDB = null;
        if (user != null && !"".equals(user)) {
            userDB = getUser(user);
        }
        if (userDB == null) {
            return "Sai tên đang nhập hoặc mật khẩu";
        }
        if (userDB.getPassword()!= password) {
            return " Sai tên đang nhập hoặc mật khẩu ";
        }
        return "Success";
    }
```

.Net:

```
private void createNewSessionId()
{
    Session.Clear();
    Session.Abandon();
    Session.RemoveAll();
    if(Request.Cookies["ASP.NET_SessionId"]!=null)
    {
        Response.Cookies["ASP.NET_SessionId"].Value = String.Empty;
            Response.Cookies["ASP.NET_SessionId"].Expires=
DateTime.Now.AddMonths(-20);
    }
    SessionIDManager Manager = new SessionIDManager();
```

```
    String NewID = Manager.CreateSessionID(Context);
    HttpCookie cookie = new HttpCookie("ASP.NET_SessionId");
    cookie.HttpOnly = true;
    cookie.Value = NewID;
      Response.Cookies.Add(cookie);
     //HttpContext.Current.Request.Cookies.Add(cookie);
}
```

PHP:

```
<?php
session_start();
session_regenerate_id();
echo session_id();
?>
```

10. Sử dụng cookie an toàn.

- **Nguy cơ:**
  - Khi người dùng không thiết lập thuộc tính "HTTPOnly" cho session cookie. Hacker có thể sử dụng mã javascript để đánh cắp session cookie của người dùng.
  - Với các ứng dụng web có sử dụng HTTPS cho các chức năng quan trọng như đăng nhập, sử dụng HTTP cho các chức năng khá, nhưng không thiết lập thuộc tính "secure" cho session cookie thì hacker nghe lén đường truyền để lấy cắp session cookie của người dùng, phiên làm việc của người dùng vẫn không được bảo vệ bởi cơ chế mã hóa của HTTPS.
- **Phòng chống**:
  - Yêu cầu thiết lập thuộc tính "HTTP Only" cho session cookie. Vì nếu Session cookie không được set thuộc tính "HTTP Only". Ta có thể cấu hình bên phía Webserver.

Ví dụ: Đối với tomcat, sửa file */tomcatdir/conf/context.xml*, thêm thuộc tính "useHttpOnly=true" như sau:

```
<Context useHttpOnly="true">
            ...
</Context>
```

- *Code java để thiết lập httpOnly:*

```
response.setHeader("SET-COOKIE" , "JSESSIONID=" + sessionid + ";HttpOnly");
```

- *Code .NET để thiết lập httpOnly:*

```
HttpCookie cookie = new HttpCookie("ASP.NET_SESSIONID");
cookie.HttpOnly = true;
```

- *Hàm PHP để thiết lập httpOnly:*

```
bool setcookie($name,$value,$expire,$path,$domain,$secure,$httponly);
```

- Với các website sử dụng HTTPS cho các chức năng quan trọng, HTTP cho các chức năng thông thường, thì cần thiết lập thuộc tính "secure" cho các session cookie để đảm bảo session cookie này được truyền qua kênh truyền có mã hóa.

Ví dụ: Đối với tomcat, sửa file */tomcatdir/conf/server.xml*, thêm thuộc tính "secure=true" vào khai báo kết nối HTTP như sau:

```
<Connector port="8080" protocol="HTTP/1.1"
   connectionTimeout="20000"
   redirectPort="8443" secure="true"/>
```

- *Code java thiết lập thuộc tính secure cho session cookie:*

```
response.setHeader("SET-COOKIE", "JSESSIONID=" + sessionid + "; secure");
```

- *Code .NET thiết lập thuộc tính secure cho session cookie:*

```
HttpCookie cookie = new HttpCookie("ASP.NET_SESSIONID");
cookie.Secure = true;
```

- *Hàm PHP thiết lập thuộc tính secure cho session cookie:*

```
bool setcookie($name,$value,$expire,$path,$domain,$secure,$httponly);
```

11. *Chuyển hướng và chuyển tiếp thiếu thẩm tra (Unvalidated Redirects and Forwards)*

- **Nguy cơ:** Hacker có thể lừa người dùng redirect đến URI có nhiễm mã độc để cài phần mềm độc hại, hoặc lừa nạn nhân khai báo mật khẩu, hoặc những thông tin nhạy cảm khác.

Ví dụ: http://www.example.com/redirect.jsp?url=evil.com ở đây evil.com chính là trang hacker muốn lừa người dùng chuyển đến.

- ***Phòng* chống:**

- Hạn chế sử dụng việc chuyển hướng và chuyển tiếp đến URI khác.
- Nếu sử dụng thì nên hạn chế truyền tham số là trang sẽ chuyển hướng đến mà nên fix trang sẽ được tham số đến.

- Nếu yêu cầu bắt buộc phải truyền tham số trang redirect thì tham số cần phải được kiểm tra tính hợp lệ của nó thông qua whitelist các trang hợp lệ.

Ví dụ: Trước khi chuyển tiếp hệ thống tới một url khác cần kiểm tra url đó có hợp lệ không, nên tạo một whitelist các url hợp lệ:

```
If (checkUrl(actionURL)) response.sendRedirect(actionURL);
Else  response.sendRedirect(urlError);
private boolean checkUrl(String url) {
        String[] urls = {"site1.com.vn"," site1.com.vn ","site1.com.vn"};
        if (urls.containsValue(url)) {
            return true;
        } else {
            return false;
        }
    }
```

12. Để lộ dữ liệu của hệ thống

- **Nguy cơ:** Trên hệ thống tồn tại các chức năng cho phép kết xuất dữ liệu truy vấn, kết quả làm việc ra dưới dạng các file excel tuy nhiên chức năng này có các lỗ hổng sau có thể làm lộ dữ liệu của hệ thống:

- File excel được lưu trữ trong thư mục con của thư mục ứng dụng.
- File excel sau khi được người dùng tải về không được xóa.
- Cho phép truy cập trực tiếp vào các file excel này mà không qua xác thực.

  Ví dụ: Những đường link sau sẽ cho phép tải một file excel mà hệ thống đã xuất ra trước đó về mà không cần qua các bước xác thực:

  http://victim.com:8080/ams/share/report_out/expDMHangHoa20121022090909.xls

- **Phòng chống:** Các dữ liệu này lưu trong thư mục bên ngoài thư mục cài đặt web server, việc thực hiện download các dữ liệu này phải qua bước xác thực và tham số phải mã hóa.

Ví dụ: Việc download file phải thực hiện qua action, việc kiểm tra này được kiểm tra quyền qua passport.

Cấu hình trong struts

```
<action    name="download"    class="com.demo.fwtest.database.DAO.Download"
method="downloadFile">
```

```
        <result type="stream" name="download">
              <param name="contentType">application/octet-stream</param>
              <param name="inputName">inputStream</param>
              <param name="bufferSize">1024</param>
        </result>
</action>
```

Action download

```
Public String downloadFile(){
        try{
              RequestParameterObject rpo=QuereyCrypt.decrypt(getRequest());
              String filename =
FileUtils.getSafeFileName(rpo.getParameter("filename"));
              File file = new
File(getRequest().getRealPath("/download/"+filename));
              inputStream = new FileInputStream(file);
              HttpServletResponse response = getResponse();
              response.setHeader("Cache-Control","no-cache");
              response.setHeader("Content-Disposition","attachment;
filename=\"" +filename+ "\"");
              response.setHeader("Pragma","public");
              response.setHeader("Expires","0");
              response.setHeader("Content-Transfer-Encoding", "binary");
              response.getOutputStream().flush();
              }catch(Exception ex){
                    return "error";
              }
              return "download";
}
```

Gọi đến action download file từ file JSP kèm tham số tên file.

```
document.location.href="$OcontextPath}/download.do?${filename}";
```

13. Thất thoát thông tin do kiểm soát lỗi và ngoại lệ không tốt

- **Nguy cơ:** Việc hiển thị chi tiết và quá nhiều thông tin lỗi khi xử lý, các thông tin này rất có ích cho hacker. Hacker có thể dựa vào các thông tin này để đoán biết hệ thống cũng như tiếp cận, khai thác lỗ hổng ứng dụng.

- **Phòng chống:**Yêu cầu tất cả các ngoại lệ đều phải được xử lý, và được lưu vào trong hệ thống log để được xử lý sau này.

Hạn chế hiện thị chi tiết miêu tả lỗi ra phía người dùng cuối chỉ nên thông báo lỗi đơn giản nhất có thể.

```
public void prepara(){
	try{
		//code
	}catch(Exception ex){
		Logger.getLogger(GridDAO.class.getName()).log(Level.SEVERE,
null, ex);
	{
}
```

Trong struts2 ta có thể cấu hình trong struts.xml

```
<result name="exception">
	share/error.jsp?errorCode=exception
</result>
</global-results>
	<exception-mapping exception="java.lang.Exception"
result="exception"/>
</Global-exception-mappings>
```

Sửa lỗi đối với việc cố tình tạo trang lỗi(Analysis of Codes) => có thể biết phiên bản chính xác của web server.

Thêm những dòng cấu hình này vào trong web.xml của ứng dụng, sau đó thêm file notfound.jsp.

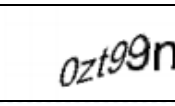
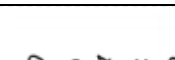
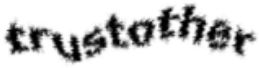
```
<error-page>
	<error-code>404</error-code>
	<location>/WEB-INF/jsp/pages/NotFound.jsp</location>
</error-page>
<error-page>
	<error-code>500</error-code>
	<location>/WEB-INF/jsp/pages/NotFound.jsp</location>
</error-page>
```

Sử dụng Captcha an toàn.

- **Nguy cơ:** Với các chức năng quan trọng, hoặc có thể lộ thông tin, hacker có thể sử dụng công cụ tự động cố gắng thực thi chức năng đó nhiều lần với các tham số khác nhau cho đến khi đạt được ý đồ của hacker.
- **Phòng chống:** Sử dụng captcha an toàn theo *Chỉ thị sử dụng Captcha an toàn* do Tập đoàn ban hành. Và việc kiểm tra captcha của 1 chức năng phải thực hiện trước khi phần chính của chức năng thực thi. Captcha có thể không sử dụng ngay khi

thực hiện chức năng mà kích hoạt sau khi có dấu hiệu nghi ngờ có tác động do công cụ tự động. Ví dụ như: Chức năng đăng nhập thực hiện thất bại 5 lần liên tiếp, có khả năng bị bruteforce mật khẩu, lúc đó hệ thống sẽ yêu cầu người dùng cần nhập captcha.

- **Một số tiêu chí captcha mạnh**

STT	Chính sách	Ghi chú
Các tiêu chí cơ bản		
1.	Mã Captcha được sinh ra một cách ngẫu nhiên.	
2.	Độ dài Captcha được sinh ngẫu nhiên không được cố định.	
3.	Kích thước các kí tự trong mã Captcha được sinh ngẫu nhiên.	
Các tiêu chí chống nhận dạng		
4.	Blurring: Làm mờ các kí tự Captcha. ***(Khuyến nghị sử dụng, không bắt buộc)***	
5.	Distortion: Làm biến dạng các kí tự Captcha.	
6.	Rotation: Xoay các kí tự Captcha luân phiên theo các góc nghiêng khác nhau.	
7.	Waving: Hiển thị mã Captcha dưới dạng lượn sóng.	
8.	Fonts: Sử dụng ngẫu nhiên nhiều loại font trong mã Captcha.	
Các tiêu chí chống phân đoạn		
9.	Background: Sử dụng hình nền gây nhiễu cho mã Captcha. ***(Khuyến nghị sử dụng, không bắt buộc)***	
10.	Lines: Sử dụng ”lines” đè qua các kí tự Captcha gây nhiễu cho mã Captcha. ***(Khuyến nghị sử dụng, không bắt buộc)***	
11.	Collapsing: Các kí tự Captcha phải dính liền, đổ nghiêng, chồng chéo lên nhau.	
Các tiêu chí hỗ trợ người dùng		
12.	Tính năng refresh hỗ trợ người dùng chọn mã Captcha mới khi gặp mã Captcha khó.	
13.	Charset: Không sử dụng tập các kí tự Captcha phức tạp, nhập nhằng.	Ví dụ: các chuỗi ký tự dễ nhầm lẫn ’l’, ’1’, ’I’; ’W’, ’w’: giống với ’v’ hoặc

		’vv’; ’O’, ’0’, ’Q’

- **Một số thư viện captcha an toàn**

STT	Thư viện	Link tham khảo
Java		
1.	Jcaptcha	http://jcaptcha.sourceforge.net/
2.	Cage - CAptcha GEnerator Java Library	http://akiraly.github.com/cage/
PHP		
3.	Securimage	http://www.phpcaptcha.org/
4.	BotDetect Captcha	http://captcha.biz
5.	freeCap	http://www.puremango.co.uk/2005/04/php_captcha_script_113/
NET		
6.	BotDetect Captcha	http://captcha.biz

14. File inclusion.

- **Nguy cơ:** Khi lập trình PHP có thể từ file hiện tại gọi đến file khác thông qua các lệnh như include, require, require _ once, include _ once. Từ đó dẫn đến nguy cơ nếu không kiểm soát tốt file được gọi đến thì hacker có thể chuyển hướng lời gọi đến các file chứa lệnh độc hại, khi đó ứng dụng sẽ gọi đến và chạy các lệnh hacker mong muốn. Hoặc hacker có thể lợi dụng để đọc các file bất kỳ của ứng dụng.
- **Phòng chống:**
  - Thiết lập websever an toàn bằng cách phân quyền cho các thư mục hợp lý, tránh ứng dụng có thể truy cập vào các file không cần thiết. Không cho phép include remote file nếu không cần thiết bằng cách thiết lập cấu hình trong file php.ini như sau:

```
allow_url_fopen=Off
```

  - Hạn chế cho phép người dùng điều khiển file include. Nếu phải thực hiện thì sử dụng đường dẫn tuyệt đối khi include file, nên sử dụng whitelist chứa danh sách các file được phép include. Ví dụ như sau:

```
<?php
$whitelist = array('index.html', 'downloads.html', 'info.html');
$page = $_GET['page'];
if(in_array($page,$whitelist)){
include($page);
}else{
```

```
die("Attack attempt");
}
?>
```

15. Command injection.

- **Nguy cơ:** Khi ứng dụng cho phép người dùng đưa dữ liệu vào các câu lệnh thực thi trên hệ điều hành, hacker có thể lợi dụng để chèn các ký tự đặc biệt cho phép nối, thực thi nhiều câu lệnh khác của hệ điều hành.
- **Phòng chống:** Validate chặt chẽ dữ liệu người dùng gửi vào, tránh các ký tự cho phép nối thêm các câu lệnh thực thi của hệ điều hành. Sử dụng whitelist là danh sách chứa các ký tự được phép xuất hiện trong dữ liệu, để so sánh, đối chiếu loại bỏ các ký tự không nằm trong danh sách đó.

Ví dụ: Loại bỏ các ký tự không nằm trong whitelist: a-z0-9\-\. Với các ngôn ngữ như sau:

- PHP

```
$param=escapeshellarg($param);
system("nslookup $param");
```

Hoặc

```
$param=preg_replace("/[^a-z0-9\-\.]/i", "", $param);
system("nslookup $param");
```

- PERL

```
$param=~s/[^a-z0-9\-\.]//gi;
$param="\"".$param."\"";
system("nslookup ".$param);
```

- ASP.NET

```
<asp:RegularExpressionValidator runat="server" id="ParamValidator"
ControlToValidate="param" ErrorMessage="Invalid input. You are allowed to
enter characters and digits only" ValidationExpression="^[a-zA-Z0-9\-\.]"
/>
```

- ColdFusion

```
<cfscript>
param=ReReplace(param,'[^a-z0-9\-\.]','','all');
</cfscript>
```

- Python

```
param = re.sub("[^a-zA-Z0-9\-\.]+", "_", param)
```

- JAVA/JSP

```
param.replaceAll("[^a-z0-9\-\.]","");
```

16. Xml/Xpath injection.

- **Nguy cơ:**
  - Khi dữ liệu từ người dùng đưa vào để phân tích, xử lý xml có nguy cơ hacker đưa vào các dữ liệu có chứa các thẻ xml gây ra mất tính đúng đắn của dữ liệu.
  - Khi dùng xpath để thao tác dữ liệu xml, lập trình viên thường sử dụng cách cộng xâu Input từ người dùng, các câu truy vấn này có thể bị mắc lỗi Xpath Injection, bằng việc lợi dụng lỗi này, kẻ tấn công có thể xem, thêm, sửa, xóa dữ liệu trong file xml từ đó chiếm được thông tin người dùng, ứng dụng, ...
- **Phòng chống:**
  - Escape hoặc encode XML dữ liệu trước khi đưa vào file dữ liệu dạng XML, để tránh hacker có thể chèn dữ liệu có chứa thẻ xml gây ra sai lệch dữ liệu trong file xml.
  - Dữ liệu input từ người dùng phải được tham số hóa trước khi đưa vào thư viện xử lý Xpath.

Ví dụ: Hàm kiểm tra input trước khi đưa vào thư viện xử lý Xpath:

```
public boolean checkValueForXpathInjection(String value) {
        boolean isValid = true;
        if ((value != null) && !"".equals(value)) {
                String xpathCharList = "()='[]:,*/ ";
                String decodedValue = URLDecoder.decode(value,
Charset.defaultCharset().name());
                for (char c : decodedValue.toCharArray()) {
                        if (xpathCharList.indexOf(c) != -1) {
                                isValid = false;
                                break;
                        }
                }
        }
        return isValid;
}
protected void doGet(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
  try {
```

```
        if
(!checkValueForXpathInjection(request.getParameter("employeeID"))) {
                response.sendError(HttpServletResponse.SC_BAD_REQUEST);
                // Trace injection
                // Exit
                return;
        }

        .....

        }
  } catch (Exception e) {
        response.sendError(HttpServletResponse.SC_BAD_REQUEST);
        e.printStackTrace();
  }
}
```

17. Các lỗi liên quan đến xử lý luồng nghiệp vụ, logic.

- **Nguy cơ:** Khi lập trình viên không xử lý đúng luồng nghiệp vụ của ứng dụng, xử lý sai về logic, có thể gây ra các lỗi mà kẻ tấn công có thể lợi dụng khái thác, tấn công chiếm quyền điều khiển ứng dụng. Ví dụ: Khi xử lý sai luồng nghiệp vụ logic kẻ tấn công có thể lợi dụng tấn công (Spam SMS, Viết chương trình tự động đăng ký dịch vụ, tài khoản....)
- **Phòng chống:** Lập trình viên khi xây dựng, phát triển ứng dụng cần lắm rõ luồng nghiệp vụ, logic của ứng dụng, xử lý các case mà kẻ tấn công có thể lợi dụng thực hiện viết chương trình tự động để thực hiện.

  Ví dụ: Một người dùng thành thạo ứng dụng có thể thực hiện đăng ký 2 khách hàng/ 1 phút, nhưng nếu thực hiện đăng ký 10 khách hàng / 1 phút là bất thường, lập trình viên có thể thiết lập captcha… để xử lý ngăn chặn các tools tự động.