



Secure Programming

Continuous Assessment 1

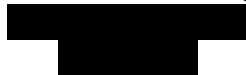
Secure programming

Stephen [REDACTED]

2018

Dean - B0009

Department of Informatics
School of Informatics and Engineering



**BN311 Bachelor of Science in Computing
Digital Forensics & Cyber Security
Secure Programming
28/10/2018**

Plagiarism Declaration

[REDACTED]
DEPARTMENT OF INFORMATICS

LECTURER [REDACTED]

DECLARATION ON PLAGIARISM

I declare that the work I/We am(are) submitting for assessment by the Institute examiner(s) is entirely my(our) own work, except where the author or source has been duly referenced and attributed.

I confirm that this material has not been previously submitted for a degree or any other qualification at [REDACTED] or any other institution. I further confirm that I have read and understood the Institute policy on plagiarism in assignments and examinations (3AS08.doc) and that I am not, so far as I am aware, in breach of any of these regulations.

Signed:

Dean,

Date: 28/10/2018

Contents

Plagiarism Declaration	2
Executive Summary	5
1 - Cross-Site Scripting-Send Message	7
Exploitation	7
Impact	7
Remediation.....	7
2 - Insecure Direct Object Referencing- Documents	9
Exploitation	9
Impact	9
Remediation.....	10
3 - Directory Transversal Attack – Download2.jsp.....	13
Exploitation	13
Impact	13
Remediation.....	13
4 - Non Secure HTTP – Website	16
Exploitation	16
Impact	16
Remediation.....	16
5 - Local file inclusion - Website	18
Exploitation	18
Impact	18
Remediation.....	18
6 - Cross Site Scripting on Login.jsp	19
Exploitation	19
Impact	19
Remediation.....	19
7 - Server Version leak	20
Exploitation	20
Impact	20
Remediation.....	20
8 – Cross- Frame Scripting (XFS) – Login.jsp	22
Exploitation	22
Impact	22
Remediation.....	22
9 – Plain text password logins -.....	24

Exploitation	Exploitation of this vulnerability was unnecessary as the dangers were obvious, so it was immediately patched.....	24
10 – Plain Low Hanging Fruit Vulnerabilities -	29

Executive Summary

This report documents the investigation into the provided Sitting Duck web application provided. It lists the investigation methods, resulting vulnerabilities found and remediation attempts to remove, obfuscate or repair any discovered flaws. The scope of the manual investigation was limited to our current working knowledge of Java, SQLI, and XSS vulnerabilities due to time restrictions. Where possible we did attempt to go outside of our comfort zone and learn to fix vulnerabilities we have not yet covered.

Both manual attempts and automated tools were used to try gain as much information, gain database access, redirect and forward links as well as a visual scan for depreciated code, poorly secured functions and any non-scripting related out of the box vulnerabilities.

Introduction

The first step in our investigation was to perform our manual investigations, both visual and active attacks. A cursory inspection of the website source code for depreciated code was undertaken. No results were found here, allowing us to move on to more aggressive methods. The next step was to begin to perform basic level URL attacks to attempt to inject SQLI attacks, XSS attacks and URL redirects. We discovered some vulnerabilities in these areas which shall be documented below. Finally, we attempt to use SQLI injection attacks on the web application login. This attack was unsuccessful. It was noted during the manual process that errors did arise which caused webpage errors to be displayed revealed information pertaining to the technologies deployed within the application, which is itself a minor vulnerability. This necessitated the addition of tries and catches to handle these errors.

It was felt at this point that we have exhausted our current working knowledge of vulnerabilities and detecting any more manually would require too much time and exceed the scope of our investigation, we therefore employed automated tools to speed up the detection of as many unknown vulnerability types.

The first tool used was 'Net Sparker'. This is a very aggressive scanner which confirmed our earlier vulnerabilities as well as highlighting some repetition of these vulnerabilities we have failed to discover. Most importantly a wealth of vulnerabilities was noted that far exceeded our known list of potential vulnerabilities.

Finally, to reconfirm our results and to see if any further vulnerabilities could be located using differing algorithms, a second tool was deployed against the web application. For this wave of attacks, 'Owasp Zap' was selected. This tool was chosen for its high detection rate in the Owasp top 10 list of vulnerabilities in particular.

Untitled Session - sitting ducks - OWASP ZAP 2.7.0

File Edit View Analyse Report Tools Online Help

Standard Mode

Sites + Quick Start Request Response +

Contexts

- Default Context
- Sites
 - http://127.0.0.1:8080

Header: Text Body: Text

HTTP/1.1 200 OK
Server: GlassFish Server Open Source Edition 4.1.1
X-Powered-By: JSP/2.3
Content-Type: text/html; charset=UTF-8
Date: Wed, 07 Nov 2018 14:54:39 GMT
Content-Length: 1575

```
<tr><td>UserName: </td><td><input type="text" name="username" value="" /></td></tr>
<tr><td>Password :</td><td><input type="password" name="password" value="" /></td></tr>
<tr><td><input type="submit" name="Login" value="Login"/></td></tr>
</table>
</form>
</div><script>alert(1);</script><div>
<div id="footer">
```

History Search Alerts Output +

Alerts (8)

- Cross Site Scripting (Reflected)
- Path Traversal
- SQL Injection - MySQL (9)
- Buffer Overflow (3)
- X-Frame-Options Header Not Set (212)
- Cookie No HttpOnly Flag
- Web Browser XSS Protection Not Enabled (214)
- X-Content-Type-Options Header Missing (214)

Cross Site Scripting (Reflected)

URL: http://127.0.0.1:8080/SittingDuck/login.jsp?err=%3C%2Fdiv%3E%3Cscript%3Ealert%281%29%3B%3C%2Fscript%3E%3Cdiv%3E

Risk: High

Confidence: Medium

Parameter: err

Attack: </div><script>alert(1);</script><div>

Evidence: </div><script>alert(1);</script><div>

CWE ID: 79

WASC ID: 8

Source: Active (40012 - Cross Site Scripting (Reflected))

Description:

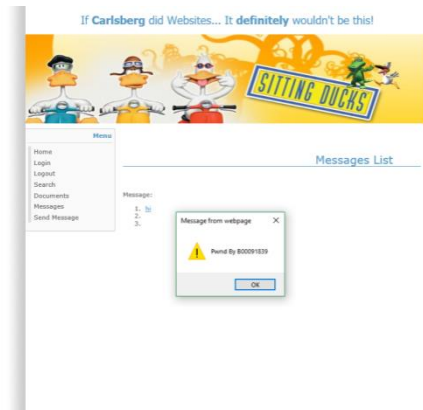
Cross Site Scripting (XSS) is an attack technique that involves convincing browser supplied code into a user's browser instance. A browser instance can be a standard web browser client, or a browser object embedded in a software product such as the browser within WinAmp, an RSS reader, or an email client. The code itself is usually written in HTML/JavaScript, but may also extend to VBScript, ActiveX, Java, Flash, or any other browser-supported technology. When an attacker gets a user's browser to execute his/her code, the code will run within the security context (or zone) of the hosting web site. With this level of

Alerts 3 2 3 0 Current Scans 0 0 0 0 0 0 0 0

ZAP By OWASP detection results

1 - Cross-Site Scripting-Send Message

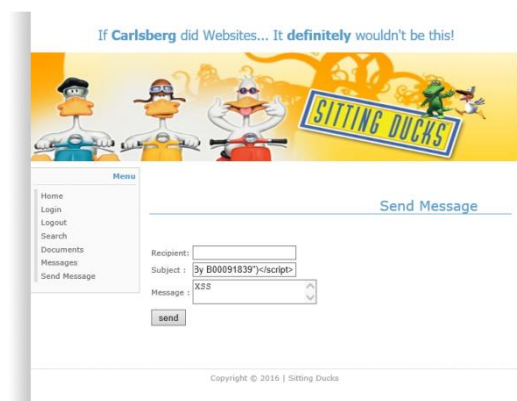
Also known as XSS, the attack uses a flaw within the code that allows a user to inject code into the working page for example a JavaScript script like an alert to create a popup.



Exploitation

XSS in the message system, it was first finding out what each part of the message does as can be seen by the hi message above, then a test of the storage by seeing what was displayed and what wasn't.

The attack was done by entering the following in the subject box which gave a XSS vulnerability
`<script>alert("Pwnd By B00091839")</script>`



This is an exploitation of the following code section.

Impact

This could allow attackers to capture Session Data, attack users with a phishing attack or intercept data via a MITM "man in the middle" attack

Remediation

The output of the input should be encoded according to the output of the location and context; this can be done using a encoding library such as OWASP ESAPI.

```
out.print("<li><a href='DisplayMessage.jsp?msgid="+rs.getString("msgid")+" '>"+rs.getString("subject")+"</a></li>");
```

Here, the subject variable is unfiltered allowing XSS attacks in the originating input.

```
<dependency>
  <groupId>org.owasp.esapi</groupId>
  <artifactId>esapi</artifactId>
  <version>2.1.0</version>
</dependency>
```

Esapi dependency added to facilitate Filtering of XSS.

```
String a = "<li><a href='DisplayMessage.jsp?msgid=";
String b = rs.getString("msgid");
String c = " '>";
String d = rs.getString("subject");
String e = "</a></li>";

out.print(a);
%>
<c:out value = "<%=b%>"/>
<%
out.print(c);
%>
<c:out value = "<%=d%>"/>
<%
out.print(e);
```

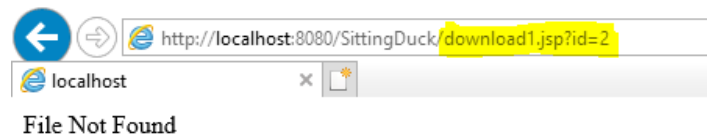
User input validated with special characters filter, input is read as string only.

2 - Insecure Direct Object Referencing- Documents

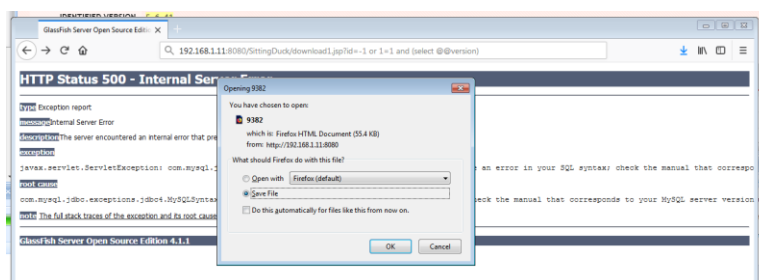
Also known as IDOR this attack allows users to bypass authorizations and access resources directly by modifying the file identification value in the URL. This is caused by the fact that the application takes a user given input and uses it to retrieve an object without doing sufficient authorization checks.

Exploitation

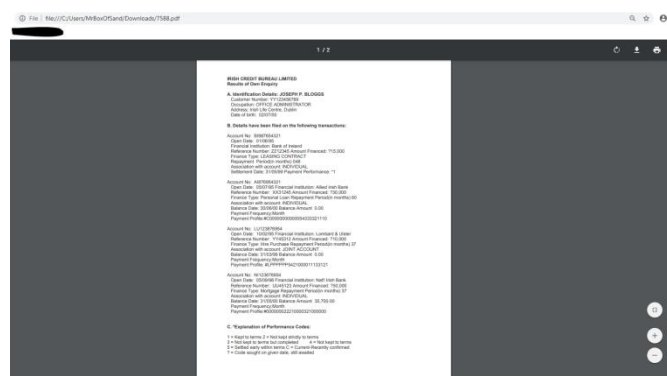
By modification of the id number it could be possible to request other id files from the web server



While trying to do a SQL injection to get the Version information of the MySQL I gave a SQL injection in the URL that looked like this 192.168.1.11:8080/SittingDuck/download1.jsp?=-1 or 1=1 and (select @@version) this didn't go as planned instead dumped out a PDF document which falls into this category



Which contained the following information this document contains banking information.



Impact

This could allow an attacker to get access to documents that otherwise required authentication to access. This vulnerability can give an attacker access to information on a system such as Database entries of files and system configurations.

The fix is to obfuscate the file id for the download in the URL so that an attacker cannot modify the ID number. This necessitates the creation and implementation of a file mapping system. Instead of using external .jsp files for downloading the documents (download1.jsp, and download2.jsp)



```

AccessReferenceMap map = null;

//If the filesMap object has not been created, create it.
if(session.getAttribute("filesMap")==null)
{
    map = FilesMap.getFilesMap();
    session.setAttribute("filesMap",map);
}
else
{
    map = (AccessReferenceMap)session.getAttribute("filesMap");
}

if(request.getParameter("file")!=null)
{
    String context = request.getContextPath();

    int BUFSIZE = 4096;
    String filePath;
    filePath = request.getParameter("file");
    filePath = (String)map.getDirectReference(filePath);
    File file = new File(getServletContext().getRealPath(context));
    file = new File(file.getParent()+"/documents/"+filePath);
    int length = 0;
    ServletOutputStream outStream = response.getOutputStream();
    response.setContentType("text/html");
    response.setContentLength((int)file.length());
    String fileName = (new File(filePath)).getName();
    response.setHeader("Content-Disposition", "attachment; filename=\"" + fileName + "\"");

    byte[] byteBuffer = new byte[BUFSIZE];
    DataInputStream in = new DataInputStream(new FileInputStream(file));

    while ((in != null) && ((length = in.read(byteBuffer)) != -1))
    {
        outStream.write(byteBuffer,0,length);
    }

    in.close();
    outStream.close();
}

```

IDOR with File mapping system implemented.

```

<hr/>
<div id="content">

    <h1>Document Download</h1>
    <ul>
        <li><a href="documents.jsp?file=<%=map.getIndirectReference("ICB.pdf")%>"> ICB.pdf </a></li>
        <li><a href="documents.jsp?file=<%=map.getIndirectReference("06171142.pdf")%>"> 06171142.pdf </a></li>
    </ul>
</div>
<div id="footer">

    <hr />
    Copyright © 2005 | All Rights Reserved

</div>

```

Indirect References for the documents

```
package com.sittingducks.files;
import org.owasp.esapi.reference.RandomAccessReferenceMap;
import java.util.HashSet;
import java.util.Set;
import org.owasp.esapi.AccessReferenceMap;

/**
 *
 * @author Cathal
 */
public class FilesMap
{
    public static AccessReferenceMap getFilesMap()
    {
        Set fileSet = new HashSet();
        fileSet.add("ICB.pdf");
        fileSet.add("06171162.pdf");

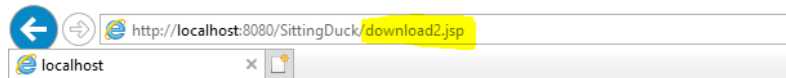
        return new RandomAccessReferenceMap(fileSet);
    }
}
```

File mapping system.

3 - Directory Transversal Attack

Directory Transversal or Path Traversal is a HTTP attack that allows attackers to access restricted directories or executes commands outside of the web servers root directory.

Exploitation



The exploitation of this vulnerability gives the attacker the ability to just attempt to alter the URL to find different locations. In this example, it is just adding a different location to the URL and seeing if it appears in this example it was download2.jsp which turned out to be a real location.

Impact

This can allow an attacker to gain unauthenticated access to files or controls that are not normally allowed to regular users.

Remediation

These pages should require a user to be authenticated for example using a cookie or token to grant authorization to the location (we selected cookies). Also, we have obfuscated the links to make it far more difficult to manually enter links in the URL. We attempted to completely disable this ability by randomly obfuscating the links such that they could not be guessed, however we ran into a small issue which prevented us from perfecting this fix in time. Lastly, we added a secure webpages folder to further obfuscate the URL link. This prevents user from simply adding /members.jsp for example to the URL and directly accessing this page.

```
String urlList[] = {"/index.jsp", "/login.jsp", "/search.jsp", "/documents.jsp", "/Messages.jsp", "/SendMessage.jsp", "/SecureWebPages/members.jsp"};

try {
    if(request.getParameter("location")!=null)
    {
        int id = Integer.parseInt(request.getParameter("location"));

        switch(id)
        {
            case 894: id = 0 ;
                break;

            case 349: id = 1 ;
                break;

            case 156: id = 2 ;
                break;

            case 147: id = 3 ;
                break;

            case 258: id = 4 ;
                break;

            case 369: id = 5 ;
                break;

            case 753: id = 6 ;
                break;

        }

        //Forwarding
        RequestDispatcher dispatcher = getServletContext().getRequestDispatcher(urlList[id]);
        dispatcher.forward(request,response);
    }
    else
    {
        out.print("Location Parameter is missing");
    }
} finally {
    out.close();
}
```

Here we see out switch, used to un-obfuscated our links

```
<div id="navigation">

    Menu

    <hr />
    <a href="<%=path%>/ForwardMe?location=894" class="navigation">Home</a>
    <a href="<%=path%>/ForwardMe?location=349" class="navigation">Login</a>
    <a href="<%=path%>/ForwardMe?location=894" class="navigation">Logout</a>
    <a href="<%=path%>/ForwardMe?location=156" class="navigation">Search</a>
    <a href="<%=path%>/ForwardMe?location=147" class="navigation">Documents</a>
    <a href="<%=path%>/ForwardMe?location=258" class="navigation">Messages</a>
    <a href="<%=path%>/ForwardMe?location=369" class="navigation">Send Message</a>

</div>
<br></br>
```

Here we see our links slightly obfuscated to make direct input more difficult.

```
pages.jsp x DisplayMessage.jsp x documents.jsp x ValidateLogin.java x members.jsp x login.jsp x SendMe
History
String user=request.getParameter("username").trim();
String pass=request.getParameter("password").trim();

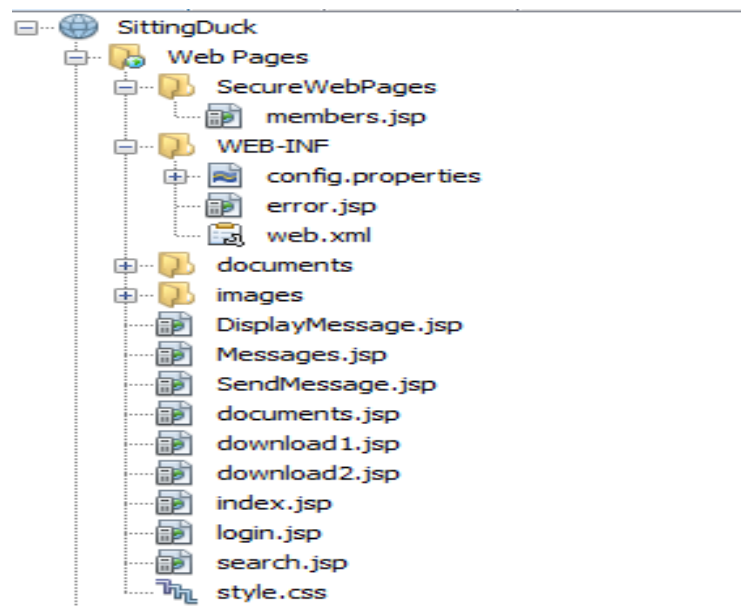
try
{
    Connection con=new DBConnect().connect(getServletContext().getRealPath("/WEB-INF/con
    if(con!=null && !con.isClosed())
    {
        HttpSession session=request.getSession();
        ResultSet rs=null;

        //NEW CODE
        PreparedStatement pst = con.prepareStatement("select * from users
        pst.setString(1, user);
        pst.setString(2, pass);
        rs= pst.executeQuery();

        session.setAttribute("user", user);
        Cookie privilege=new Cookie("privilege","user");
        response.addCookie(privilege);
        Cookie allow = new Cookie("allow","disallow");
        response.addCookie(allow);

        if(rs != null && rs.next()){
            session.setAttribute("allow","allow");
            response.sendRedirect("members.jsp");
        }
        else{
            response.sendRedirect("login.jsp?err=something went wrong");
        }
    }
}
catch(Exception ex)
```

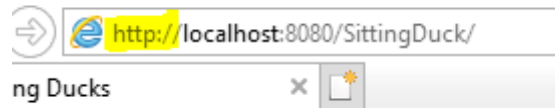
Here we have our implementation of cookies for user authentication.



Here we see our new file structure

4 - Non Secure HTTP – Website

Passwords sent over Http instead of Https



Exploitation

There are no attacks to be done for this right now, an attacker could intercept this traffic over the network and attempt to steel credentials.

This has been partially fixed since we now has our password pre validation, however, there is still one point of vulnerability between the log in page and validation page.

Impact

Passwords and confidential information could be captured over networks by attacks.

Remediation

Password hashing from input could be employed, however the best to way to fix this is would be to employ a valid SSL or TLS certificate. This was felt to be beyond the scope of the assignment, however a local certificate could be generated with a moderate amount of recoding and local generation of a certificate. Should this be deployed on a online service this certificate would need to be issues by a valid licensing body.

Getting Started

To enable HTTPS on your website, you need to get a certificate (a type of file) from a Certificate Authority (CA). Let's Encrypt is a CA. In order to get a certificate for your website's domain from Let's Encrypt, you have to demonstrate control over the domain. With Let's Encrypt, you do this using software that uses the [ACME protocol](#), which typically runs on your web host.

To figure out what method will work best for you, you will need to know whether you have [shell access](#) (also known as SSH access) to your web host. If you manage your website entirely through a control panel like [cPanel](#), [Plesk](#), or [WordPress](#), there's a good chance you don't have shell access. You can ask your hosting provider to be sure.

With Shell Access

We recommend that most people with shell access use the [Certbot](#) ACME client. It can automate certificate issuance and installation with no downtime. It also has expert modes for people who don't want autoconfiguration. It's easy to use, works on many operating systems, and has great documentation. [Visit the Certbot site](#) to get customized instructions for your operating system and web server.

If [Certbot](#) does not meet your needs, or you'd like to try something else, there are [many more ACME clients to choose from](#). Once you've chosen ACME client software, see the documentation for that client to proceed.

If you're experimenting with different ACME clients, use our [staging environment](#) to avoid hitting [rate limits](#).

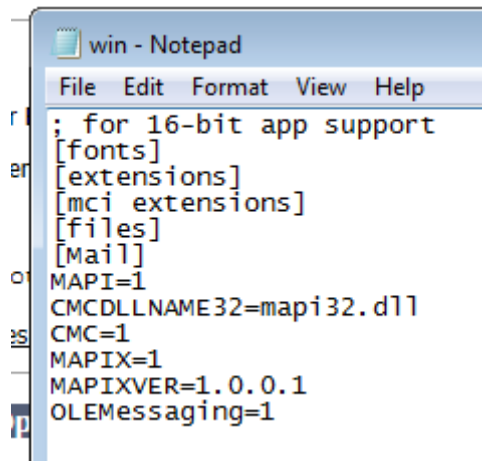
Here we see an online service used to issue SSL and TLS certificates.

5 - Local file inclusion – Website (

A local file inclusion attack occurs when a file from a target system is injected into the attacked server page, this vulnerability is based upon read permissions of the web server user.

Exploitation

<http://127.0.0.1:8080/SittingDuck/build/web/documents/../../../../../../../../windows/win.ini>



Here we see a win.ini file pulled from our system using the vulnerability.

Impact

An attacker can allow the user to grab usernames from “/etc/passwd” if available, this could also give access to areas such as the log files or the error log file. Combining this attack with upload it would be possible to upload a file or inject log files.

Remediation

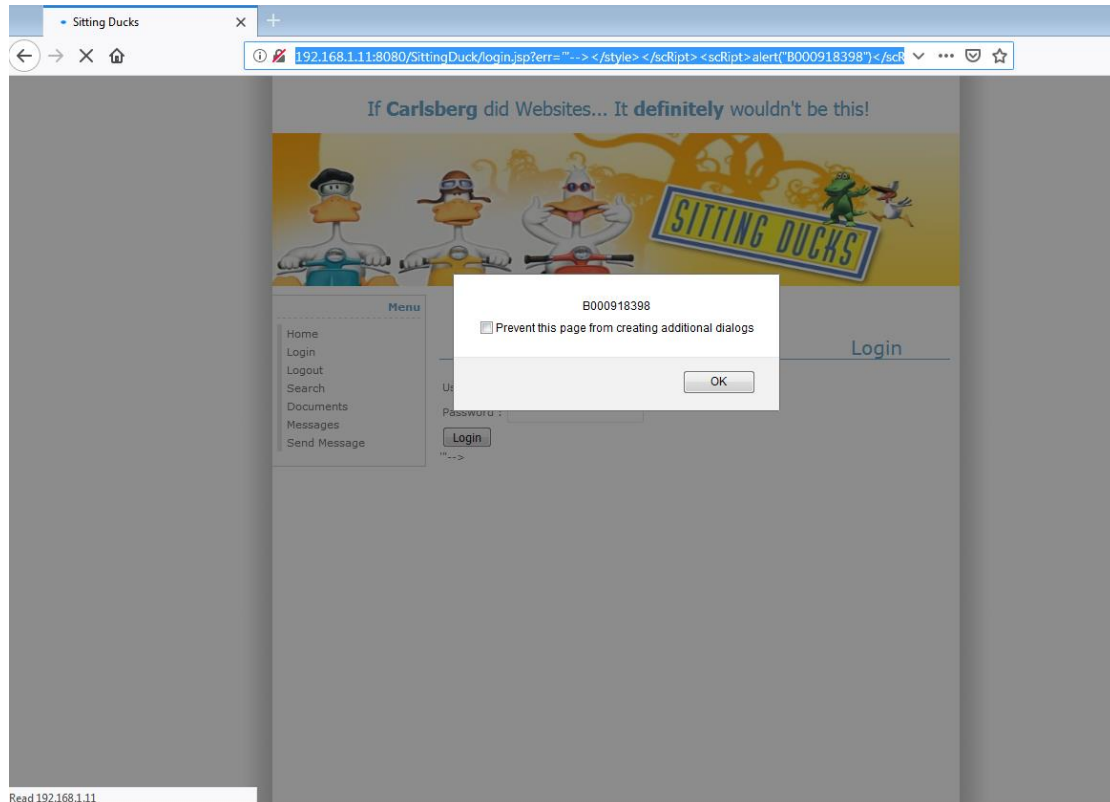
Make file paths hard-coded or selectable from limited hard-coded path lists in via and index variable, make sure to disallow characters such as “..” “/” and “%00” or any similar characters. We feel that in implementing a fix for one of our other discovered vulnerabilities this has been patched, however we are unable to identify exactly which fixed has removed this issue.

6 - Cross Site Scripting on Login.jsp

Exploitation

the attack was done by entering the following string after the URL by evoking the error response to a login after defining it as an alert,

127.0.0.1:8080/SittingDuck/login.jsp?err='</style></script><script>alert("B000918398")</script>



Impact

This could allow attackers to capture Session Data, attack users with a phishing attack or intercept data via a MITM "man in the middle" attack

Remediation

This can be fixed by using a generic error message rather than retrieving any data from the URL.

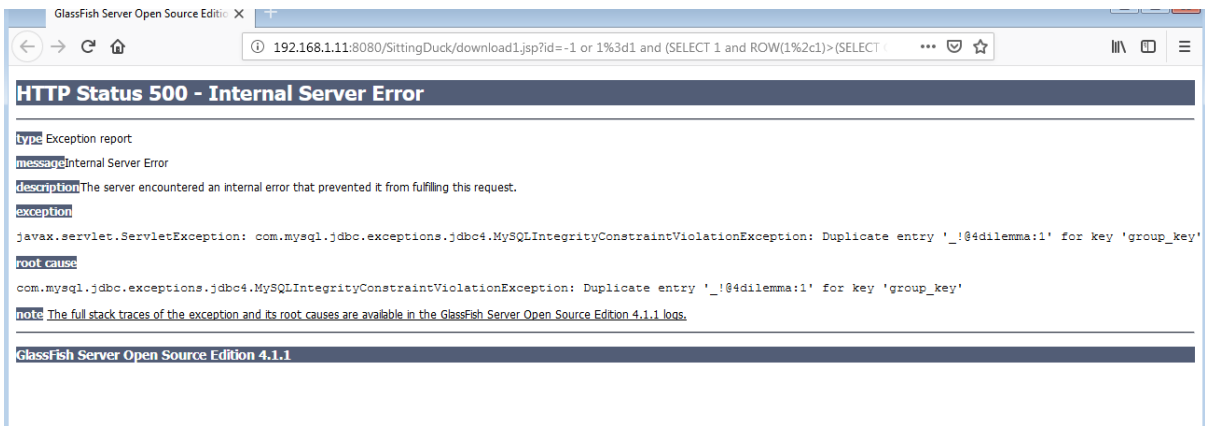
```
</form>
<%
    if(request.getParameter("err")!=null){
        out.print("Something went wrong");
    }
%>
```

Here we see our generic error message implemented.

7 - Server Version leak

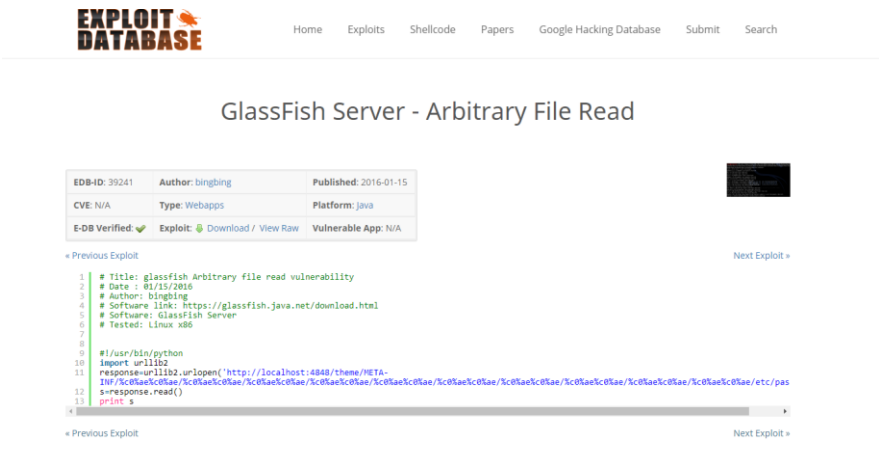
Exploitation

While working on another exploit I managed to get the server to give an error 500 page which gives to.



Impact

This could allow an attacker to find a possible vulnerability in the version if there is one released. For example, this vulnerability “Glassfish Server – Arbitrary File Read” that allows an attacker to access the Glassfish oracle admin console.



Remediation

A generic error.jsp page should be implemented to override all basic errors and negate information leakage. However, it should be noted that in Internet Explorer (the default browser for Glassfish) a lower limit of 512kb exists, so any error page must contain more then 512kb of data to be rendered.

```

<error-page>
  <error-code>400</error-code>
  <location>/WEB-INF/error.jsp</location>
</error-page>

<error-page>
  <error-code>401</error-code>
  <location>/WEB-INF/error.jsp</location>
</error-page>
<error-page>
  <error-code>403</error-code>
  <location>/WEB-INF/error.jsp</location>
</error-page>
<error-page>
  <error-code>404</error-code>
  <location>/WEB-INF/error.jsp</location>
</error-page>
<error-page>
  <error-code>500</error-code>
  <location>/WEB-INF/error.jsp</location>
</error-page>

```

Here we see our error codes added to our web.xml file

```

<%@ page isErrorPage = "true" %>
<%
    String path = request.getContextPath();
%>

<html>
<head>
  <title>Show Error Page</title>
</head>
<body>
  <h1>Oops...</h1>
  <p>Sorry, an error occurred.</p>
  <p>Please return to the <a href="<%=path%>/ForwardMe?location=894" class="navigation">Home</a> page.</p>
  <BR>

  <h1>Did you know that internet explorer blocks error pages below 512kb?</h1>
  <p>A generic error.jsp page should be implemented to override all basic errors and negate information leakage. However,
  |
  </body>
</html>

```

Here we see our error.jsp redirected implemented at the top of the page.

8 – Cross- Frame Scripting (XFS) – Login.jsp

This attack combines both malicious JavaScript and iframe, by loading a legitimate page with a script to attempt to steal data from an unsuspecting user; this type of attack is usually seen along with a social engineering

Exploitation

By modifying the request in the URL it was possible to do an IFrame injection that would allow an attacker to add malicious script to the website



Impact

This vulnerability gives an attacker the possibility to forget a link to give to a victim through social engineering that could allow for an attacker to steal data from the user.

Remediation

Make sure that only requests are allowed from trusted domains and do not accept user input for URL's unless it is really needed.

```

60
61 <h1>Login</h1>
62 <form action="ValidateLogin" method="post">
63     <table>
64         <tr><td>UserName: </td><td><input type="text" name="username" value="<%=username%>" /></td></tr>
65         <tr><td>Password :</td><td><input type="password" name="password" value="<%=password%>" /></td></tr>
66         <tr><td><input type="submit" name="Login" value="Login"/></td></tr>
67     </table>
68 </form>
69
70 <%=
71     if(request.getParameter("err")!=null){out.print(request.getParameter("err"));}
72 %>
73 </div>
74 <div id="footer">
75     <hr />
76     Copyright © 2016 | Sitting Ducks
77

```

Here we see the cause of cross frame XS on the login page.

```

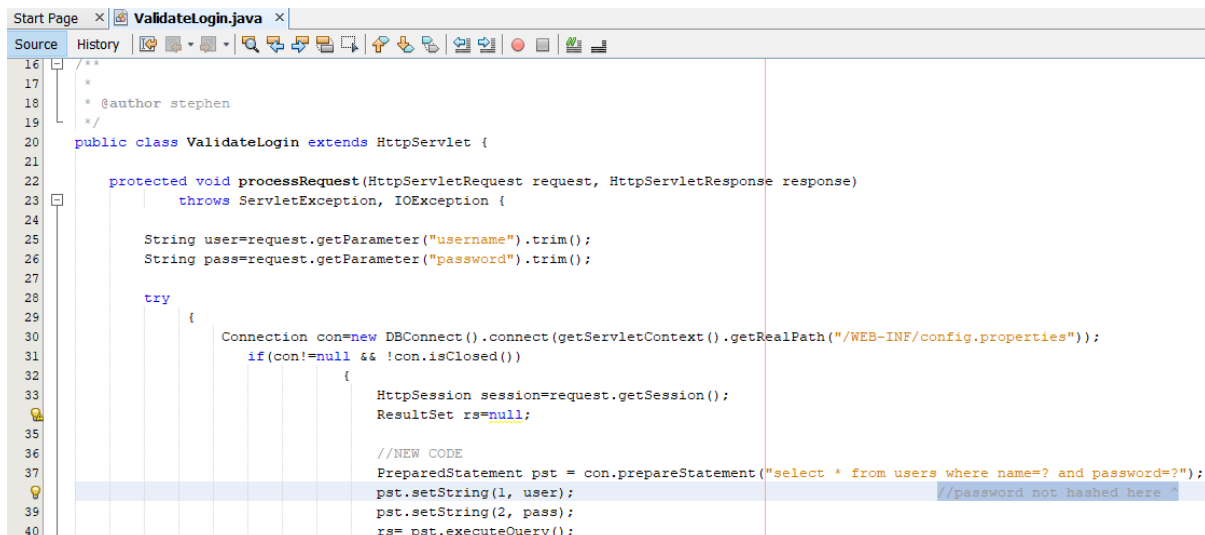
</form>
<%
    if(request.getParameter("err")!=null){
        out.print("Something went wrong");
    }
%>

```

Here we see our error message hardcoded redirecting any attempt to retrieve error data from the URL.

9 – Plain text password logins -

Passwords are entered into the database during creation in plain text. This means that passwords be accepted and validated in plain text to facilitate logins.



```
16  /**
17   *
18   * @author stephen
19   */
20  public class ValidateLogin extends HttpServlet {
21
22      protected void processRequest(HttpServletRequest request, HttpServletResponse response)
23          throws ServletException, IOException {
24
25          String user=request.getParameter("username").trim();
26          String pass=request.getParameter("password").trim();
27
28          try
29          {
30              Connection con=new DBConnect().connect(getServletContext().getRealPath("/WEB-INF/config.properties"));
31              if(con!=null && !con.isClosed())
32              {
33                  HttpSession session=request.getSession();
34                  ResultSet rs=null;
35
36                  //NEW CODE
37                  PreparedStatement pst = con.prepareStatement("select * from users where name=? and password=?");
38                  pst.setString(1, user);
39                  pst.setString(2, pass);
40                  rs= pst.executeQuery();
```

Un-hashed password validation.

```
INSERT INTO `users` (`id`, `userid`, `name`, `password`, `email`) VALUES
(1, 1, 'Fuller Ramirez', 'YZH38ZQJ4ZJ', 'sapien.Aenean.massa@tristiqueac.edu'),
(2, 2, 'Neville Mann', 'PSZ66VZM6CV', 'eu@faucibus.org'),
(3, 3, 'Talon Baldwin', 'AUH18IEB7JW', 'vestibulum.nec@Donecdignissimmagna.ca'),
(4, 4, 'Upton Dominguez', 'WOK97UIQ9BR', 'Suspendisse.non.leo@mollis.com'),
(5, 5, 'Blaze Hurley', 'WQB24XIF6IC', 'consequat.lectus@egetmetus.edu'),
(6, 6, 'Edan Hatfield', 'VFU11YUJ5YV', 'Donec.tincidunt@necmetus.com'),
(7, 7, 'Baw Santos', 'BFC77TQ84BB', 'in@non.net')
```

Un-hashed hardcoded password creation in MYSQL database.

Exploitation

Exploitation of this vulnerability was unnecessary as the dangers were obvious, so it was immediately patched.

Remediation

Passwords must be hashed in Database creation and hashed pre-validation from user input.


```

cadb
Limit to 1000 rows
62 INSERT INTO `users` (`id`, `userid`, `name`, `password`, `email`) VALUES
63 (1, 1, 'Fuller Ramirez', PASSWORD('YZH38ZQJ4ZJ'), 'sapien.Aenean.massa@tristiqueac.edu
64 (2, 2, 'Neville Mann', PASSWORD('PSZ66VZM6CV'), 'eu@faucibus.org'),
65 (3, 3, 'Talon Baldwin', PASSWORD('AUH18IEB7JW'), 'vestibulum.nec@Donecdignissimmagna.c
66 (4, 4, 'Upton Dominguez', PASSWORD('WOK97UIQ9BR'), 'Suspendisse.non.leo@mollis.com'),
67 (5, 5, 'Blaze Hurley', PASSWORD('WQB24XIF6IC'), 'consequat.lectus@egetmetus.edu'),
68 (6, 6, 'Eden Hatfield', PASSWORD('VEH11MITEM'), 'Donec.flores@egestas.com')

```

Here we see the passwords being hashed in the database

```

PreparedStatement pst = con.prepareStatement("select * from users where name=? and password = password(?)");
pst.setString(1, user);
pst.setString(2, pass);
rs= pst.executeQuery();

```

Here we see the passwords being hashed in the Web application pre validation.

10 –Buffer Over flow

Buffer Overflow is an attack where by an attacker can enter large strings of text into the URL in order to overflow a page's allowed buffer size. This can crash the server creating a server leak of information. Furthermore, this type of attack can, in extreme situations, be used to execute script.

Exploitation



Here we see an example of a buffer overflow attack being carried out on the server producing a general 500 error.

Impact

In this instance, only generic server information leakage occurred.

Remediation

The fix for this issue was to set the buffer size to a dynamic amount, controlled by the file size.

```
if(request.getParameter("file")!=null)
{
    String filePath;
    filePath = request.getParameter("file");
    filePath = (String)map.getDirectReference(filePath);
    File file = new File(getServletContext().getRealPath("/") + path);
    file = new File(file.getParent()+"/documents/"+filePath);

    int length = 0;
    ServletOutputStream outStream = response.getOutputStream();
    response.setContentType("text/html");
    response.setContentLength((int)file.length());
    String fileName = (new File(filePath)).getName();
    response.setHeader("Content-Disposition", "attachment; filename=\"\" + fileName + "\"");

    byte[] byteBuffer = new byte[(int)file.length()];
    DataInputStream in = new DataInputStream(new FileInputStream(file));

    while ((in != null) && ((length = in.read(byteBuffer)) != -1))
    {
        outStream.write(byteBuffer,0,length);
    }

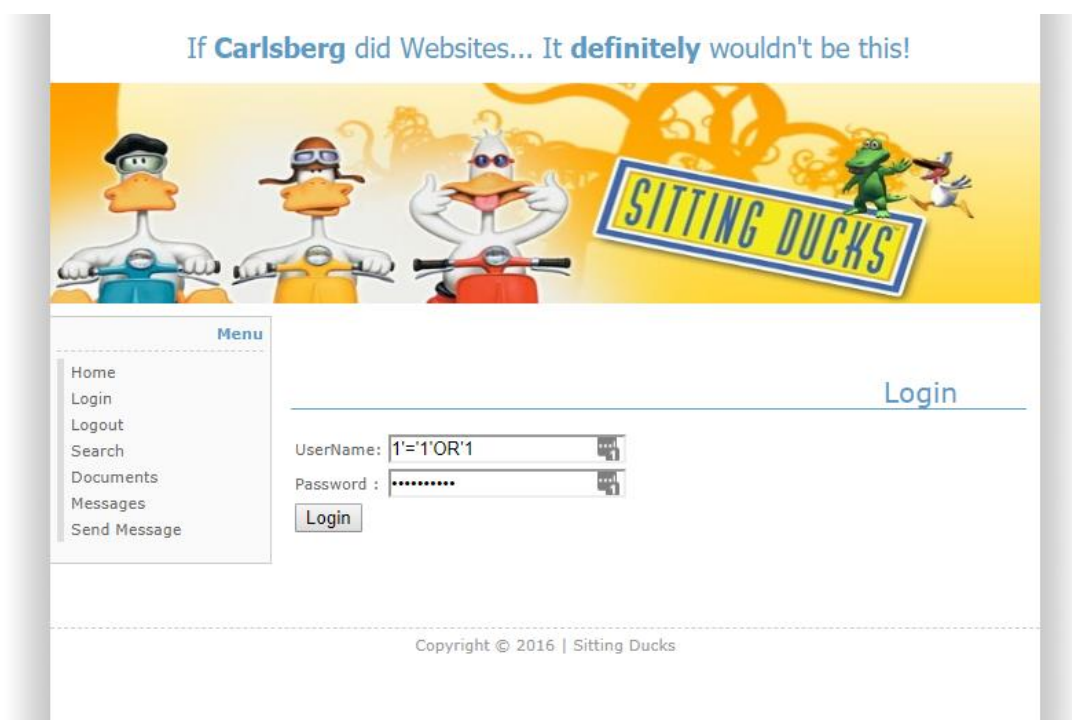
    in.close();
    outStream.close();
}
else
{
}
```

Here we see the buffer size set dynamically to file length: `byte[] bytebuffer = new byte[(int)file.length()];`

11 – SQL Injection

This is one of the more common vulnerabilities still found on most website or web applications. This form of attack is used to directly inject SQL commands into databases to retrieve restricted information/

Exploitation



Here we see an example of an SQL injection attack being performed on the web application.

Impact

This is one of the most dangerous forms attack as entire databases can be retrieved and exploited without adequate protection.

Remediation

Properly prepared statements coupled with whitelist protections in extreme cases should be implemented to SQL Injection attacks, however; in this case only prepared statements were needed.

```
Start Page x ValidateLogin.java x
Source History
16
17
18  * @author stephen
19  */
20  public class ValidateLogin extends HttpServlet {
21
22      protected void processRequest(HttpServletRequest request, HttpServletResponse response)
23          throws ServletException, IOException {
24
25          String user=request.getParameter("username").trim();
26          String pass=request.getParameter("password").trim();
27
28          try
29          {
30              Connection con=new DBConnect().connect(getServletContext().getRealPath("/WEB-INF/config.properties"));
31              if(con!=null && !con.isClosed())
32              {
33                  HttpSession session=request.getSession();
34                  ResultSet rs=null;
35
36                  //NEW CODE
37                  PreparedStatement pst = con.prepareStatement("select * from users where name=? and password=?");
38                  pst.setString(1, user);
39                  pst.setString(2, pass);
40                  rs= pst.executeQuery();
```

Here we see the implementation of a prepared statement to negate an SQL attack.

12 – Plain Low Hanging Fruit Vulnerabilities -

It was felt that to list and fix all low hanging fruit vulnerabilities would exceed the scope of this report, therefore we decided to list them only to show that they had been noted. They were as follows:

- SameSite Cookie not implemented: This would potentially allow for CSRF information leakage. This could be fixed by implanting 'Set-Cookie: key=value; SameSite=strict' into the Set-Cookie header.
- Missing X-Frame-options Header: This places the web application at risk of a click-jacking attack. Essentially this could allow an attacker to imbed a hiding overlaying page which is mostly transparent and contains a direct or download link which appears to be part of the original page but is in fact over-laid over it. This can be fixed by sending the correct X-Frame-Options in the HTTP response headers
- Missing Content Security Policy: This facilitates easier XSS scripting on the webpage, while it is true to say that XSS can still happen with a CSP declared. This can be fixed simply by declaring the meta tag '`<meta http-equiv="Content-Security-Policy" content="script-src 'self';">`' or 'content-Security-Policy: script-src 'self'' in a response header.
- Referrer-policy not Implemented: This is a request header that when not implanted may allow for cross domain leakage. This can be fixed by adding '``' to a response header or implementing the meta tag '`<meta name="Referrer-Policy" value="no-referrer | same-origin"/>`'.
- Version Disclosure (Java Servlet): This information could be used to harvest specific vulnerabilities pertaining to the version of the java servlet deployed. This can be fixed by configuring the web server to prevent information leakage for the X-Powered-By header of its HTTP response.
- Auto-complete: Auto complete is Turned on all user inputs. Of particular note is the password autocomplete. This was fixed by adding 'autocomplete="off"' to the form declarations of the username and password fields. The rest were not altered.
- Failure to mark Cookies as HTTP only: When cookies are marked as HTTP only they cannot be read client-side scripts. This provides an additional layer of XSS protection. This can be fixed by marking Cookies as HTTP only, however it should be noted that this is not a bullet proof solution as tools such as XSS 'Tunnel' can be used to bypass HTTP protection. It will protect against those less determined or less skilled attempting to attack the web application.
- Database Error Message Disclosure: As mentioned earlier any information leaked pertaining to technologies deployed on the web application could be used to data mine vulnerabilities which could then be used. This can be fixed by creating either an if statement to wrap

around any code blocks that would screen against invalid input or creating a generic error page that would be displayed through any catch handling error messages.

- **Database Version Out-Of-Date:** The current version of MYSQL deployed on our test setup is out of date. Generally, this mean that there will be more vulnerabilities' available and security updates will not have been designed with older systems in mind. This could be fixed simply by making sure that the database software is always running the latest version. However, due to potential compatibility issues this will not be done in this case.
- **Options Method Enabled:** This method leaks information on technologies deployed on the system. This can be fixed by disabling the Options Method in production systems.

Breakdown of Workload:

Conclusions

Many other potential minor code vulnerabilities where discovered which given a broader scope we would have mentioned. For example, it was felt that the ability of visitors not logged in to leave a message which was displayed as a null sender was a low vulnerability leading to potential XSS attempts due to the fact that is a indicating the database is seeking a value for this variable. This we patched, however; we did not list this fix as we felt we had potentially exceeded the brief.

Overall, we found this assignment to be quite challenging yet extremely beneficial to our understanding of secure programming practices and most importantly; thinking ahead when developing such code to be implemented. The group appreciated the opportunity to dive into such a worthwhile assignment.