

**EE6052 Distributed System Group Project Report**

**Jobmarket Online Application Security Test Report**

**Group Member**

**Table of Content:**

**Chapter 1 A practical security test how to tested the security of your application ---------------------- 1**

**Chapter 2 A discussion that establishes the following for each Top 10 item ------------------------------ 9**

**Chapter 3 Group member Contribution Summary ------------------------------------------------------------ 17**

**Chapter 1 - A practical security test, where you outline how you tested the security of your application**

In this web application security test, we use the OWASP Zed attack agent (ZAP) as the attacking environment, which is one of the most popular free security auditing tools in the world and is actively maintained by hundreds of international volunteers. It helps you to automatically find security vulnerabilities in Web applications as you develop and test your applications.

The ZAP mainly used for penetration testing. It is very easy to use. The steps for testing are shown as the followings:

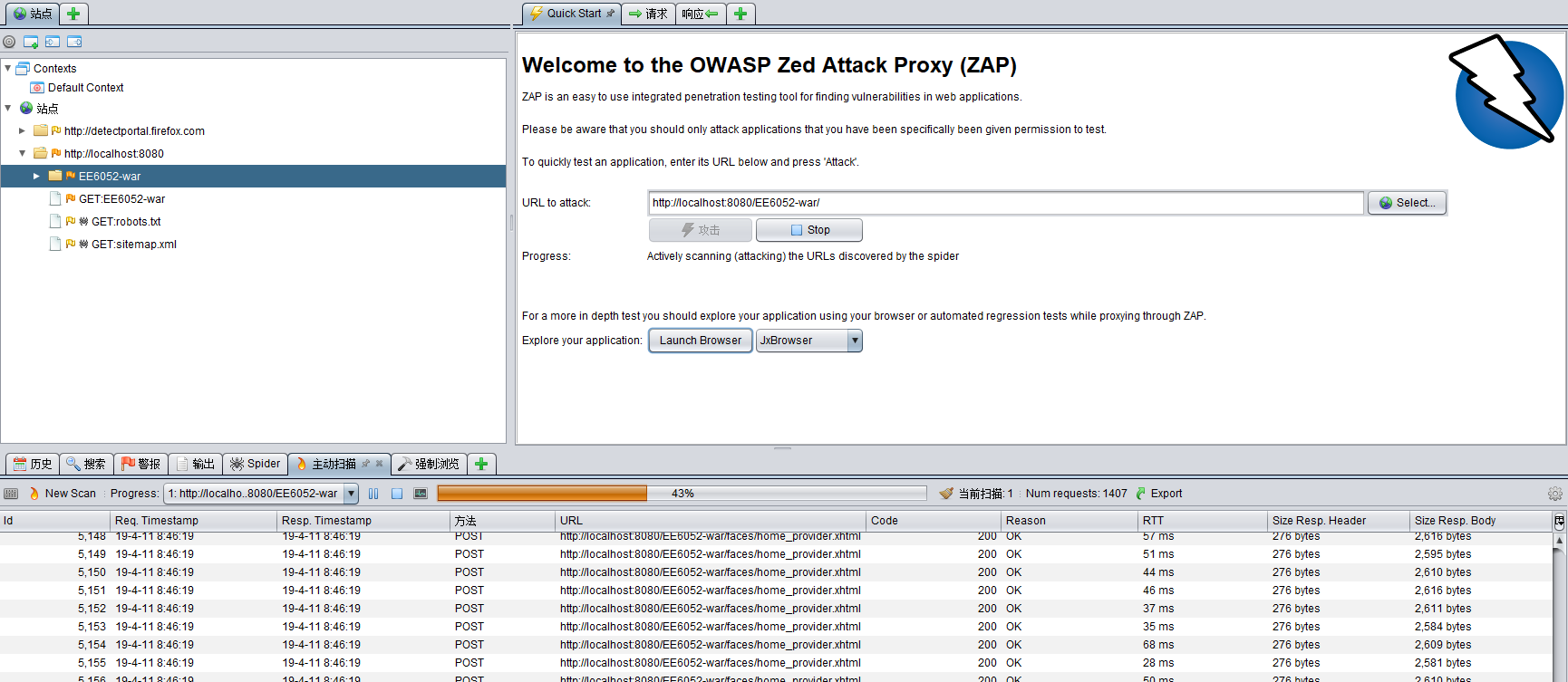
**Step 1: Configure browser proxy as ZAP.**

Open a browser and go to the network settings. Change the proxy settings to host=localhost, port=8085. That is the application network address for ZAP. Also, make sure that the proxy is not bypassed for local addresses.

In this way, we can grab any HTTP via the browser. And we can modify and resend the request.

**Step 2: Start a quick web security testing**

As show the following, input the URL of our webapp into the testing target input field and click attack button to start the testing.

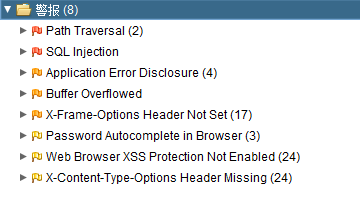


The whole testing is based on the web crawler automation. Firstly, crawlers crawl all pages of the site under test. And then, passively scan all acquired pages during page fetching. After that, the page, functions and parameters are analyzed by means of active scanning.

**Step 3: Get alarm result, analysis and justify its existence**

After the qick scanning we can get the security alarm on the left connor of the ZAP application. There are two high-level alarms, three medium alarms and two low alarms.

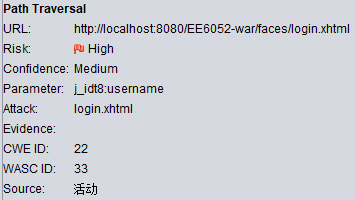
As the testing is very basic scanning test, which may just provide the alarm as long as it has the possibility of the vulnerability existence. Thus, we need go into the further step for each alarm to justify its existence at the step 4.



**Step 4: Justify each vulnerbility on the website.**

1. **Path Traversal**

The Path Traversal attack technique allows an attacker access to files, directories, and commands that potentially reside outside the web document root directory. An attacker may manipulate a URL in such a way that the web site will execute or reveal the contents of arbitrary files anywhere on the web server. Any device that exposes an HTTP-based interface is potentially vulnerable to Path Traversal.

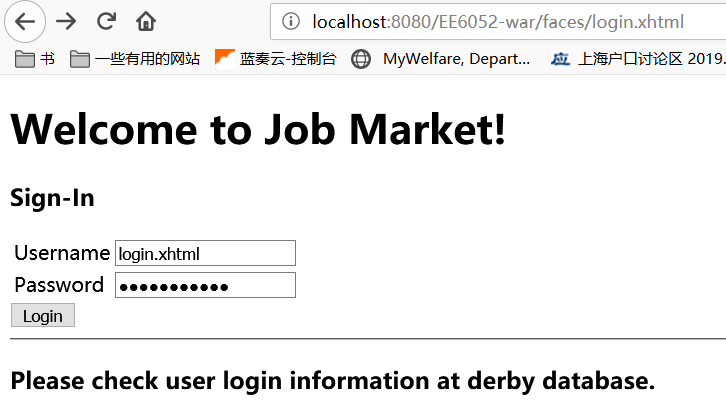


But the concrete attack should be varified here.

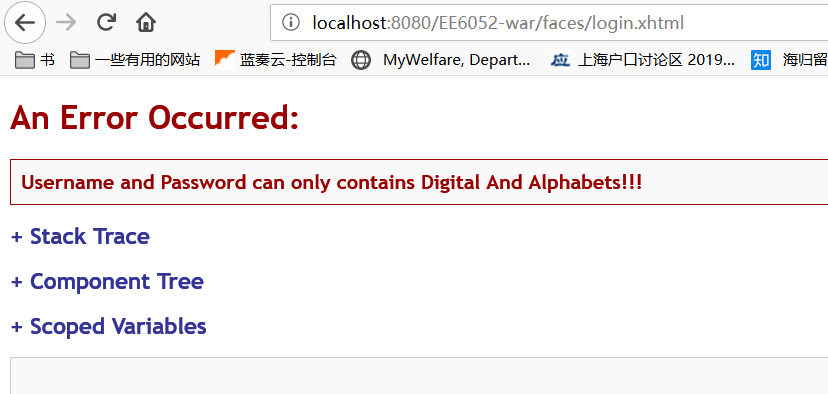
*Login in the website http:localhost: 8080/EE6052-war/faces/login.xthml*

*And input the tested parameter at the input field of ‘username’ with value of ‘login.xhtml’*

*Then, click login button.*



*And we can see the website throws a UnsupportedOperationException as the following:*



***We can see that the ZAP software mis-considerated the HTTP respond as a anti-expect result and categorized as an vulnerable problem. In fact, it is a validator module which is a part of the business logic at the website.***

**!Notice! The real factor resulting in Path Traversal is not the parameter problem shown aboved on the ZAP attack simulation. But a loophole vulnerability for glassfish server, which will be detailed on the following:**

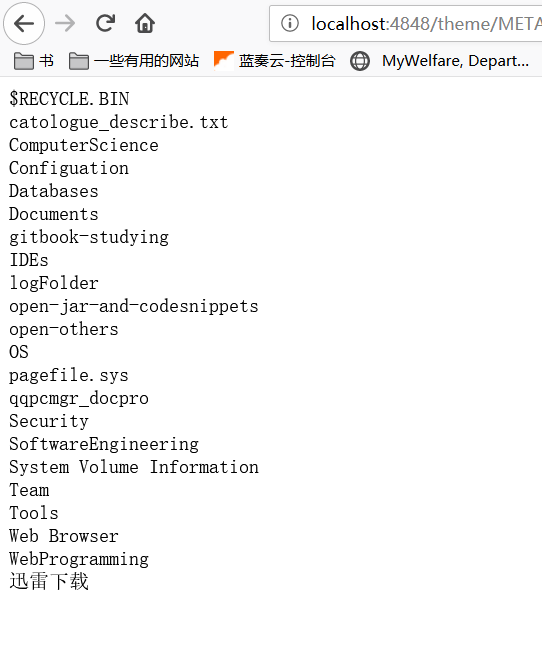
|  |  |
| --- | --- |
| [*CVE-2017-1000028*](http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2017-1000028) | *Oracle, GlassFish Server Open Source Edition 4.1 is vulnerable to both authenticated and unauthenticated Directory Traversal vulnerability, that can be exploited by issuing a specially crafted HTTP GET request.* |

This vulnerability can both found at glassfish server 4.1 at administrator webroot page and web application page. Its implementation can be shown as the following statement:

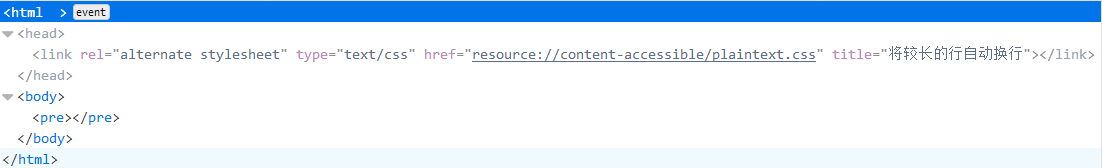
**http://localhost:4848/theme/META-INF/****%c0%ae%c0%ae/%c0%ae%c0%ae/%c0%ae%c0%ae/%c0%ae%c0%ae/%c0%ae%c0%ae/%c0%ae%c0%ae/%c0%ae%c0%ae/%c0%ae%c0%ae/%c0%ae%c0%ae/%c0%ae%c0%ae/**

with the URL, ‘**%c0%ae%c0%ae/’** means ‘..’ – As in Java, encoding schema is unicode, and ‘%c0%ae’will be parsed as \Uc0ae character, which is ‘.’.

The following shows the evidence of that vnlnerability – just put the above URL into the browser address input area. And it returns the root directory contents on my physical disk D:\\.



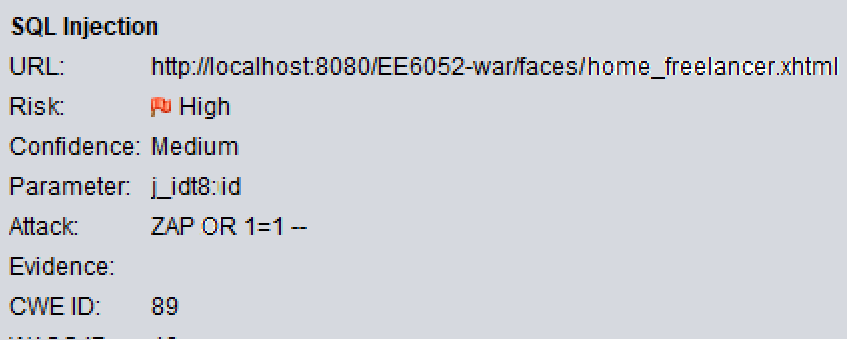
In addition, to this web job market online application (localhost:8080/EE6052-war). I input the URL into the address field as: <http://localhost:8080/EE6052-war/faces/%c0%ae%c0%ae>, it returns me the following webpage (HTML code):



In face, this webapage should display the 404 webpages as I did not create webpage at XHTML dirctory and I didn’t configure it in my web.xml file as well – this webpage not exist actually. But it does returned on browser without any reason.

**The remedy method of that will be disscussed on the next chapter A6 and A9, which is a type of system misconfiguration and the glassfish server is the component with known Vulnerabitlies.**

1. **SQLInjection**

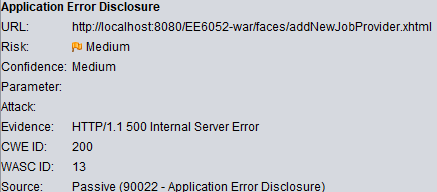
Structured Query Language (SQL) is a specialized programming language for sending queries to databases. The SQL programming language is both an ANSI and an ISO standard, though many database products supporting SQL do so with proprietary extensions to the standard language. Applications often use user-supplied data to create SQL statements. If an application fails to properly construct SQL statements it is possible for an attacker to alter the statement structure and execute unplanned and potentially hostile commands. When such commands are executed, they do so under the context of the user specified by the application executing the statement. This capability allows attackers to gain control of all database resources accessible by that user, up to and including the ability to execute commands on the hosting system. 

This part will be deliberated on the next chapter.

Please see the next chapter

1. **Application Error Disclosure**

This page contains an error/warning message that may disclose sensitive information like the location of the file that produced the unhandled exception. This information can be used to launch further attacks against the web application. The alert could be a false positive if the error message is found inside a documentation page.

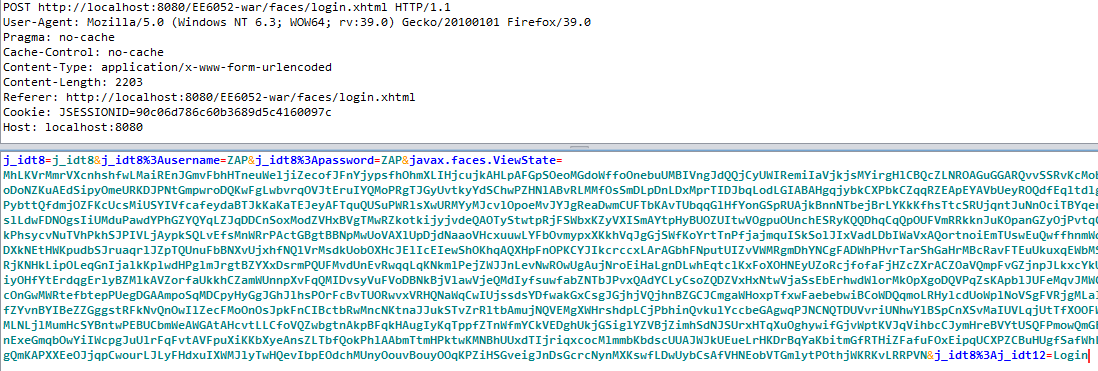


As shown the following, the webpage dispalied the Server Internal Error which should not displaied at the user’s webpage. Because user may utilize this application error to penetrate into the web application.

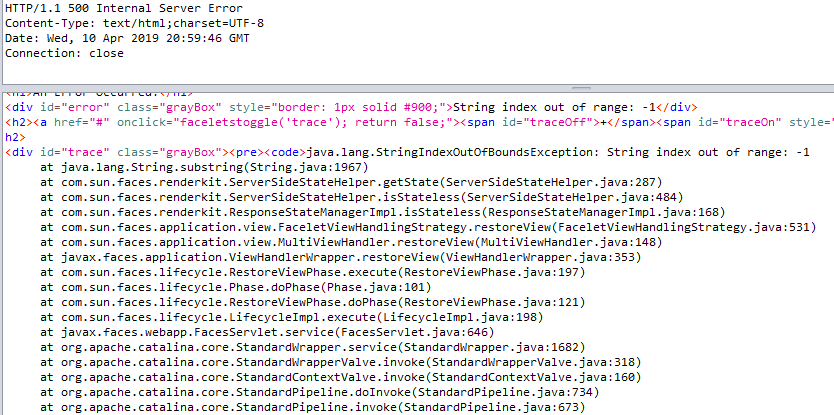


1. **Buffer Overflowed**

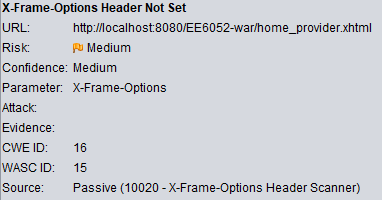
Here we can see that, we edit a POST HTTP request with very long content parameter.



When server received this HTTP POST request, it will get the StringIndexOutOfBoundsException, which means it is very possibly cause the leaking of memory, and as the end of Denial of Service (DoS).

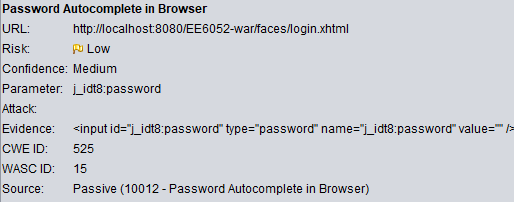


1. **X-Frame-Options Header Not Set**

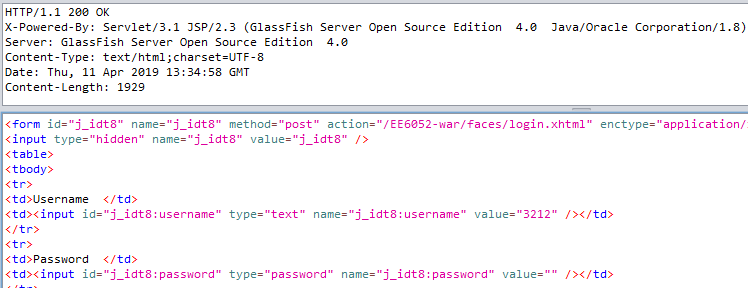


X-Frame-Options header is not included in the HTTP response to protect against 'ClickJacking' attacks.

1. **Password Autocomplete in Browser**



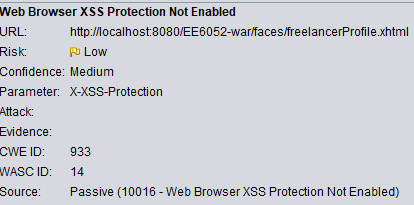
The AUTOCOMPLETE attribute is not disabled on an HTML FORM/INPUT element containing password type input. Passwords may be stored in browsers and retrieved.



However, we can see that the Evidence, <input id=…. Value=””/> the value of the input area for password is empty. I also check the respon of the HTTP request, it still is empty.

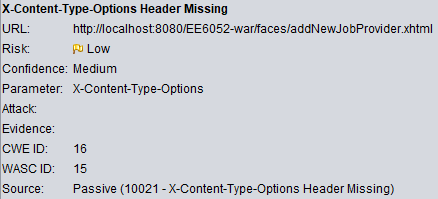
Thus, this vulnerablity is not exist.

1. **Web Browser XSS Protection Not Enabled**



Web Browser XSS Protection is not enabled, or is disabled by the configuration of the 'X-XSS-Protection' HTTP response header on the web server

1. **X-Content-Type-Options Header Missing**



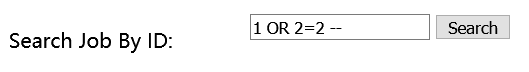
The Anti-MIME-Sniffing header X-Content-Type-Options was not set to 'nosniff'. This allows older versions of Internet Explorer and Chrome to perform MIME-sniffing on the response body, potentially causing the response body to be interpreted and displayed as a content type other than the declared content type. Current (early 2014) and legacy versions of Firefox will use the declared content type (if one is set), rather than performing MIME-sniffing.

**Chapter 2** - **A discussion that establishes the following for each Top 10 item**

1. **A1: Injection (1 mark)** 
   1. **Injection Testing Idea**

Generally, a way for injection is SQL Injection, which is a normal way for getting data from the database of website.

For testing the vnlnerability, we purpose is to implements the SQL statement “SELECT \* FROM Table(?)”, for that, we apply the well-designed SQL command so that it can meets the true condition so that all data records at the database can be returned, which shown as the followings:



Based on search functionality, we can guess the original SQL statement for searching a job by id could be like:

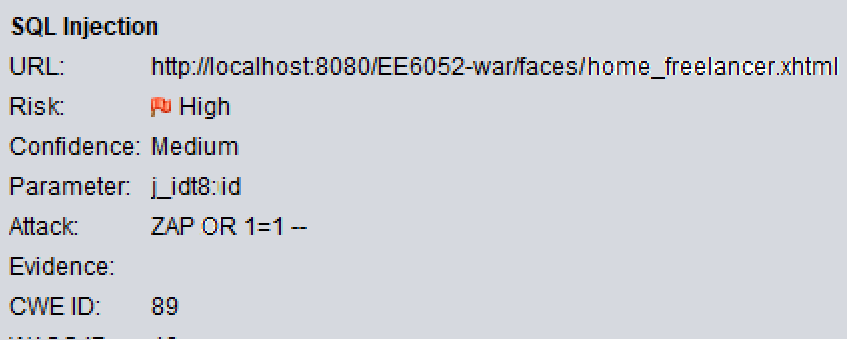
SELECT \* FROM JobTable WHERE id =? ;

Now, we insert the 1 OR 2=2 statement into the place of ‘?’, we can get: SELECT \* FROM JobTable WHERE id = 1 OR 2 =2 --;

The whene constrain on above statement is always true, thus all data at JobTable could be returned, resulting data leaking.

* 1. **Vulnerability Testing**

Step 0: run ZAP and test the webpage (home\_freelancer.xhtml) We can get the testing result from the ZAP:



Step 1: Running the website and logined as a Freelancer user type, then input the above SQL Injection value on the Search Job By Id field

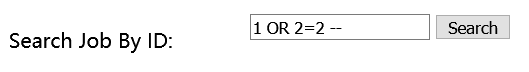
Step 2: Click the Search button

Step 3: Nothing will be return. It still on the webpage sending the form, and the input text field is empty:

C:\Users\dell\AppData\Local\Temp\1554921891(1).png

Generally, a way for injection is SQL Injection, which is a normal way for getting data from the database of website.

For testing the vnlnerability, we purpose is to implements the SQL statement “SELECT \* FROM Table(?)”, for that, we apply the well-designed SQL command so that it can meets the true condition so that all data records at the database can be returned, which shown as the followings:



Based on search functionality, we can guess the original SQL statement for searching a job by id could be like:

SELECT \* FROM JobTable WHERE id =? ;

Now, we insert the 1 OR 2=2 statement into the place of ‘?’, we can get: SELECT \* FROM JobTable WHERE id = 1 OR 2 =2 --;

The whene constrain on above statement is always true, thus all data at JobTable could be returned, resulting data leaking.

* 1. **How do we avoid SQL Injection?**

In this venture we have not utilized the previously mentioned codes that allow the aggressors to hack the records from our database, rather utilizing the Java Persistence Query Language (JPQL), which provides the functionality for SQLInjection-proof. When invalied data type received as parameters, it will returned as empty list.

***@NamedQuery(name = "Users.findByUsername", query = "SELECT u FROM Users u WHERE u.username = :username")***

***, @NamedQuery(name = "Users.findByPassword", query = "SELECT u FROM Users u WHERE u.password = :password")***

***, @NamedQuery(name = "Users.findByType", query = "SELECT u FROM Users u WHERE u.type = :type")})***

1. **A2:** **Broken Authentication**

Authentication is the process of verification that an individual, entity or website is who it claims to be.

* 1. **How to test it**

As we login to the webiste there is session storing the username as ‘3212’, when user login out, the username text input filed willl auto-complete the username value as following:



Here we wirte the following testing code for brute-force penetrating into the system:

***for(pwd=000;pwd<=9999;pwd++){  
 swipeLock();  
 pressButton();  
 if(package.status == close) continue;  
 else break;  
}***

When pwd=3212, it breaks. Then we get the password for account of ‘3212’. And authentication has been broken.

* 1. **What’s consequence**

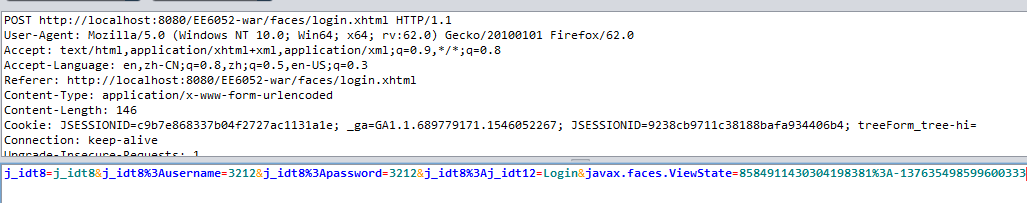
System will accept a request comes from one user, whereas in reality another user has issued the request. When the fake user login the website. The fake user can do any damage as the role of current user.

* 1. **How to repair it**

Most authentication attacks occur due to the continued use of passwords as sole factor. The website registration system should require user select a complexed password which may consisted of digital, alphabets, symbols or case sensitive any others. In addition, once we find a specific account number has been trying to log up to 5 times, it should clock the account for a time. Maybe to add CAPTCHA is also a good solution, which can stop the scirpt automation for brute-force testing.

1. **A3: Sensitive Data Exposure**
   1. **How to test it**

Open the ZAP tool and configure the proxy, when we login in the website as a freelance with account number of ‘3212’ and passsword of ‘3212’. The package of HTTP POST request is shown the following now:



We can see that the passwrod is not be hashed or encrypted during the data trafering time. It is a very serious data problem because the sensitive data is exposed at HTTP request.

1. **How to prevent it**

Encrypt all data in transit with secure protocols such as TLS with perfect forward secrecy (PFS) ciphers, cipher

prioritization by the server, and secure parameters. Enforce encryption using directives like HTTP Strict

Transport Security (HSTS).

1. **A4: XML External Entities (XXE)**

This vulnerability is applicable to the application in the sense that an untrusted data can be inserted into the XML documents.

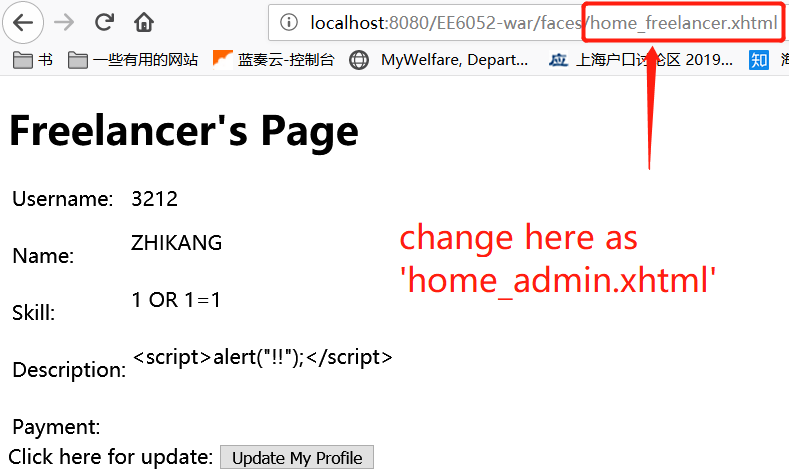
There are possibilities of an attackers exploiting vulnerable XML processors if they can upload XML or include hostile and malicious content in an XML document, exploiting vulnerable code, dependencies or integrations. This can be a hole that can be used to extract data, execute a remote request from the server, scan internal systems, perform a denial-of-service attack, and execute other attacks.

How to secure my Apps

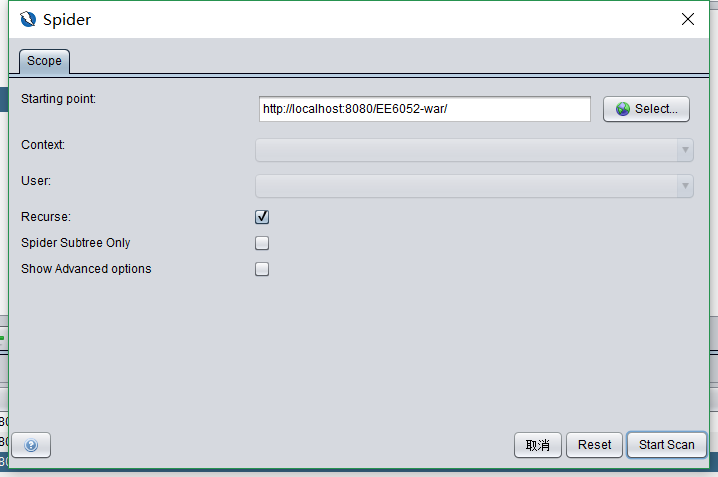
This application can be secured by constant patch or possible upgrade of all XML processors and libraries in use by the application or on the underlying operating system. Dependency checkers can be used. Also , disabling XML external entity and DTD processing in all XML parsers in the application. Implementing positive ("whitelisting") server-side input validation, filtering, or sanitization to prevent hostile data within XML documents, headers, or nodes will also be of help in filtering unwanted content.

1. **A5: Broken Access Control**
   1. **Testing steps**

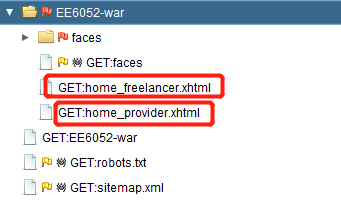
Firstly, we logined as a freelancer’s identity:



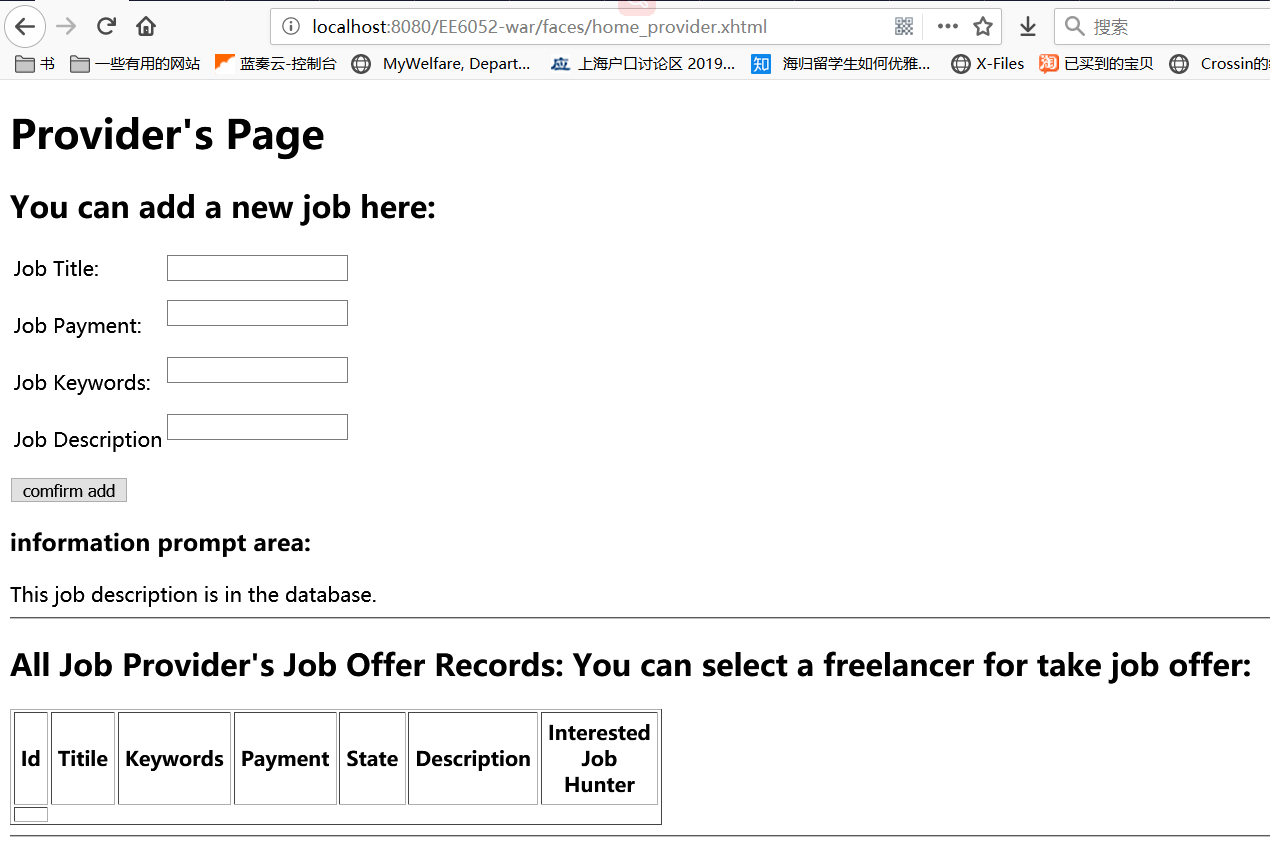
Then, we utilize the ZAP software crawler module to scan the website structure:



Then we found two more page of the website.

.

Then I input the URL address as value of: localhost:8080/EE6052/faces/home\_provider.xhtml. And we get the following webpage as return:



The page is designed for provider’s user, which means freelancer can not reach at the page.

However, we get. This is the broken access control.

* 1. **What’s the consequece**

After the evil freelance login, the webpage, they can make any changes such as deliver a new job or delete existed job cause the business logic disordered.

* 1. **How to fix the bug**

To control the user’s access previlege, we need to modify the business logic controller – checking the session for everytime user login the webpage. If the current user’s session is empty. The system will send back login.xhtml webpage to user.

In addition, in web.xml file, we can specify the URL rules for navigation, list the white-list rule for access based on user type.

1. **A6: Security Misconfiguration**

At Chapter 1, application testing, on the step 4, we found ZAP prompt it exists the path traversal vnlnerability, which caused by the old verison of glassfish server. User can use the delibertaed designed path controling statement return the physical directory content of the server host machine.

Such as the URL: <http://localhost:8080/EE6052-war/faces/%c0%ae%c0%ae>,. It will return the empty webpage.

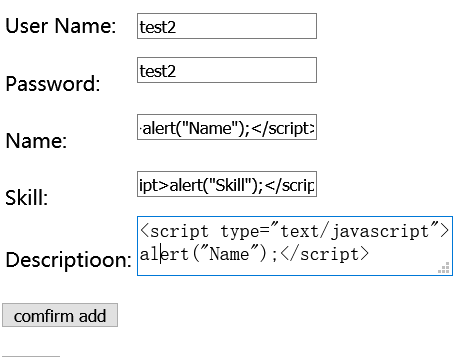
That is not what we what. We can specify the URL navigator pattern for constraining user inputed URL navigation. We can list all webpage at this web application and add them into the whitelist configuration. Any others resource request on the server will be rejected.

1. **A7: Cross Site Scripting (XSS)**
   1. **What the XSS caused?**

The essential protection against cross-site scripting, relevant yield encoding changes over show information from possibly risky code to inactive show information. It does this by changing over unique characters regularly utilized as a part of code—like the not as much as sign (<)— and replaces them with HTML encoded substances, or escape codes, that advise the framework to render those characters as content, not as code.

* 1. **Testing XSS**

At the Chapter 1 testing part, we can know that the Web Browser XSS Protection Not Enabled at the site: <http://localhost:8080/EE6052-war/faces/freelancerProfile.xhtml>. We logined as administrator and add a new freelancer, with the followings data inputed:



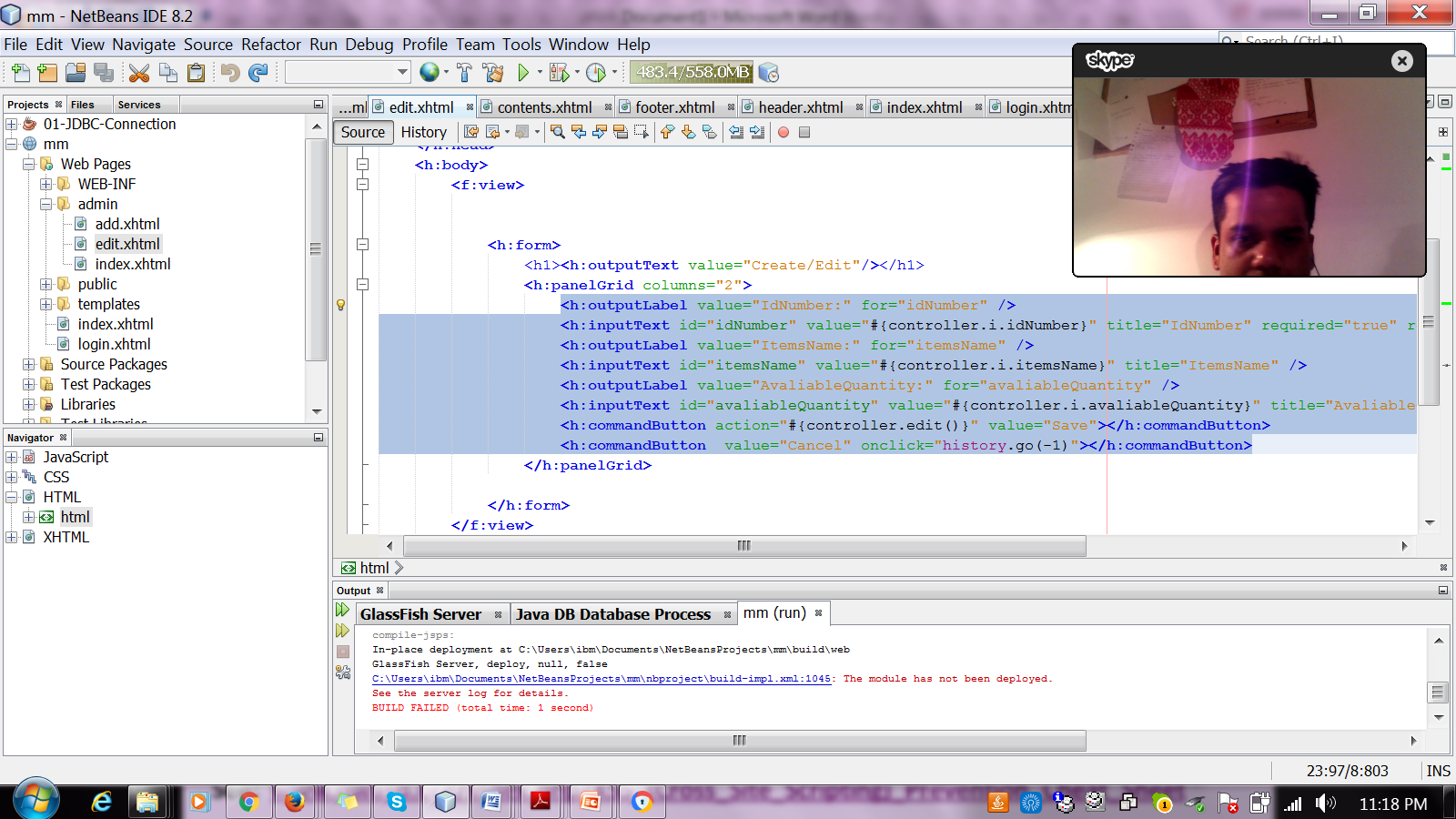
Theoritically, when the information of the freelancer displaied on webpage, it will triger the XSS attack: now let’s login in as the ‘test2’ freelancer’s identity.

However, there is no alert() prompt comes up when the webpage is loaded – It shows there is no XSS vulnerbility at this webpage.



* 1. **How we did in our Web Application**

In Java EE's MVC system JSF has builtin XSS counteractive action over all place, where it is handled by UI Component..



1. **A8: Insecure Deserialization**

The use of XML significantly reduces the likelihood of introducing insecure deserialization vulnerabilities.

An attacker can as well take advantage of inherent vulnerability if malicious content is uploaded and the applications deserialize the malicious content by an attacker.

How to secure my application

Avoid accepting serialized objects from untrusted sources, also using serialization mediums that only permit primitive data types.

1. **A9: Using Component with known Vulnerabitlies**

To the best of our knowledge this application is not vulnerable to using components with known vulnerability. The application was developed with components without known vulnerabilities.

At Chapter 1, application testing, on the step 4, we found ZAP prompt it exists the path traversal vnlnerability, which caused by the old verison of glassfish server. User can use the delibertaed designed path controling statement return the physical directory content of the server host machine.

Such as the URL: <http://localhost:8080/EE6052-war/faces/%c0%ae%c0%ae>,. It will return the empty webpage. Which is not what we what.

This situation caused by [*CVE-2017-1000028*](http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2017-1000028) *,* the application use known vunlerable component – glassfish server. There is a remedy method for this section, we can update the glassfish server so that whose version over 4.1.0.

1. **A10: Insufficient logging and monitoring**
2. **What’s Insufficient logging and monitoring?**

According to the OWASP Top 10 the bedrock of every major incident is the exploitation of insufficient logging and monitoring. Due to lack of monitoring and timely response, attackers rely on this to achieve their goals without been detected.

1. **Insufficient Logging and Monitoring Testing Idea**

Generally, a way for logging and monitoring is storing of log files.

1. In order to capture and store audit log events, the use of a separate and dedicated security-hardened server platform is used.
2. Synchronizing the systems clocks is made possible with the use of Network time synchronization technology.
3. Having a strong access control to logs
4. The implementation of an alert system for monitoring staff should be ensured for 24/7 monitoring
5. Multiple failed login attempts for system authentication and event logs is stored.
6. **Vulnerability Testing**

**Step 1**: Look over the source code, system architecture making sure routines are put in place to handle the logs from every application and system

**Step 2**: Proper routines prevents little log values.

1. **How to avoid this**

An incident response and recovery plan should be established or adopted in the source code and system architecture.

The establishment of effective monitoring and alerting system put in place to detect suspicious activities and respond in a timely fashion

Logs been generated are easily understood by a centralized log management system.

.

**Chapter 3** – **Project Group Member Contribution Summary [Important]**

**This chapter clearly reflets the contributions of each member in the group.**