FINAL REPORT

Software Testing and Quality Assurance

by
Md. Abdur Rahman
ID: 011202260
Section: A

1. Introduction

This report documents the security testing conducted on the orangehrm software with a focus on identifying vulnerabilities through various test cases. The goal of this testing is to identify different weaknesses of the software against various security attacks.

2. Tools Overview

Initially, the *OWASP ZAP* (Zed Attack Proxy) tool was selected to conducted the security testing. ZAP is a widely recognized open-source application that is used as a security scanner. It is a highly extensible tool that supports range of plugins and scripts. Primarily it includes features like:

- *Automated scanning*
- *Interception proxy*
- *Web crawler*
- *Fuzzer*
- *Token extractor*
- *Query decoder*
- *Script engines, and many more.*

However, it still lacks features to generate attacks like *MITM*, Address spoofing, *ARP poisoning*, etc. Thus for testing, another tool named *Ettercap* was selected which includes attacks such as:

- *ARP poisoning*
- *NDP poisoning*
- *DHCP spoofing*
- *Port stealing*
- *SSL intercepting, etc.*

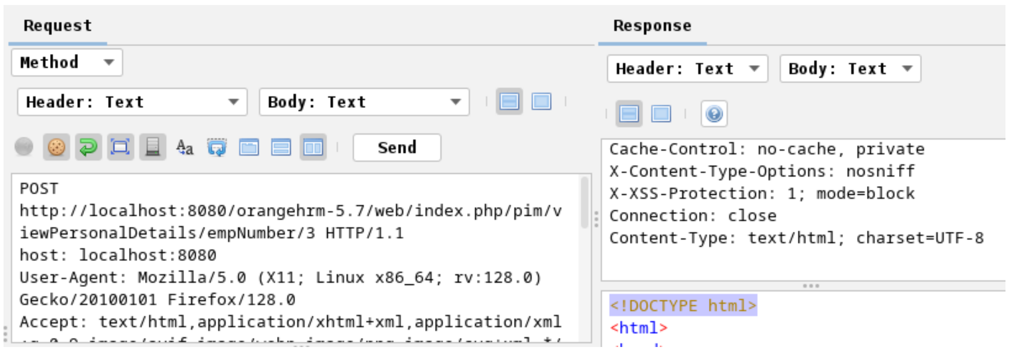
3. Test Cases Overview

A list of conducted security test cases are as listed below:

TC01. To check unauthorized update of users' info (ZAP)
TC02. To view other users' personal info as non-admin user (ZAP)
TC03. To check add employee feature with malicious entries (Manual)
TC04. To insert with improper file name to do Path Traversal Attacks (Manual)
TC05. To upload vulnerable XSS payloads in the file attachment (ZAP)
TC06. To check SQL injection vulnerability in the login functionalities (ZAP)
TC07. To check for bruteforce attempts vulnerability in the login functionalities (ZAP)
TC08. To check the access control to the users for *candidates'* application (ZAP)
TC09. To perform the purge action to all users and vacancy posts (Ettercap)
TC10. To perform the purge action to all users and vacancy posts (ZAP)
TC11. To validate input sanitization in user registration forms (Manual)
TC12. To test for session expiration after logout (ZAP)
TC13. To verify that file uploads are restricted to allowed types only (Manual)
TC14. To scan for vulnerabilities in the Buzz Module (ZAP)
TC15. To test SQL injection in the *search* functionalities (ZAP)
TC16. To check for CSRF protection on sensitive form submissions (ZAP)
TC17. To check DNS Spoofing for Redirecting Traffic (Ettercap)

4. Details on Each Test Cases

TC01: To check unauthorized update of users' info:

Goal	In PIM module, under Employee Information tab, all of the info about an employee can be seen (and as admin: can be modified). Goal is to check that by intercepting ongoing request-response action, and then modifying the request (as a non-admin user) if anyone can update other user's information.
Steps	● Start ZAP as a proxy. ● Log in as a non-admin user. ● Go to the PIM section in *orangehrm* and access the employee list. ● Capture the request to access user information in ZAP. ● Modify the request (header) to attempt an unauthorized update of another user's info. ● Make a *POST* request ● Resend the modified request. ● Observe the response in ZAP 
Expectation	Non-admin users should not be able to update any information of other users.
Observation	Non-admin users cannot update any information of other users. It is only possible by the user himself and the admin.
Risk	None

```
POST
http://localhost:8080/orangehrm-5.7/web/index.php/pim/v
iewPersonalDetails/empNumber/3 HTTP/1.1
host: localhost:8080
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:128.0)
Gecko/20100101 Firefox/128.0
Accept: text/html,application/xhtml+xml,application/xml
```

```
Cache-Control: no-cache, private
X-Content-Type-Options: nosniff
X-XSS-Protection: 1; mode=block
Connection: close
Content-Type: text/html; charset=UTF-8
```

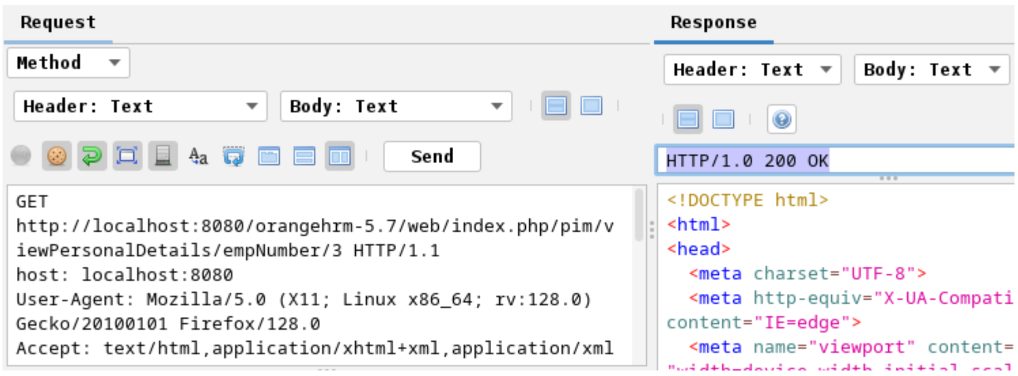
```
<!DOCTYPE html>
<html>
```

TC02. To view other users' personal info as non-admin user

Goal	This is somewhat similar to TC01, but here instead of trying to modify information (which failed), it was checked if the (private) information such as joining dates, addresses, contact numbers, or user names can be seen as a non-admin user.

Steps

- Start ZAP as a proxy.
- Log in as a non-admin user.
- Go to the PIM section in *orangehrm* and access the employee list.
- Capture the request to access user information in ZAP.
- Modify the request and make a *GET* request
- Resend the modified request.
- Observe the response in ZAP



```
GET
http://localhost:8080/orangehrm-5.7/web/index.php/pim/v
iewPersonalDetails/empNumber/3 HTTP/1.1
host: localhost:8080
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:128.0)
Gecko/20100101 Firefox/128.0
Accept: text/html,application/xhtml+xml,application/xml
```

```
HTTP/1.0 200 OK
```

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compati
content="IE=edge">
  <meta name="viewport" content=
```

Expectation

Non-admin users should not be able to view any private information of other users.

Observation

Although the response returned 200 OK, The information was not visible while logged in as a non-admin user.

Risk

None

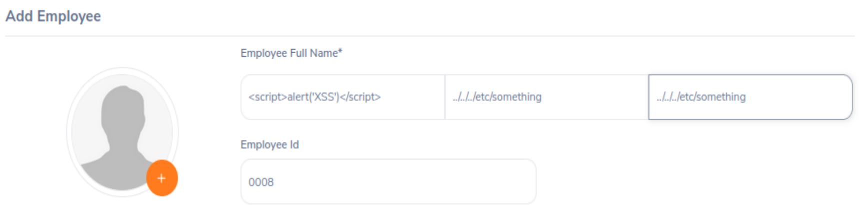
TC03. To check add employee feature with malicious entries

Goal

Here, it was check whether the system supports malicious entries as First/Middle/Last name in the *Add Employee* option. This action (adding employee) can only be done by Admin, so the goal was to check if malicious entries (or inappropriate ones) can be put in the these fields.

Steps

- Login as Admin
- Go to PIM
- Select Add Employee option
- Put malicious (or odd) code as name entries in the fields
- Try to create employee
- Check if system throws any warming.



Add Employee

Employee Full Name*

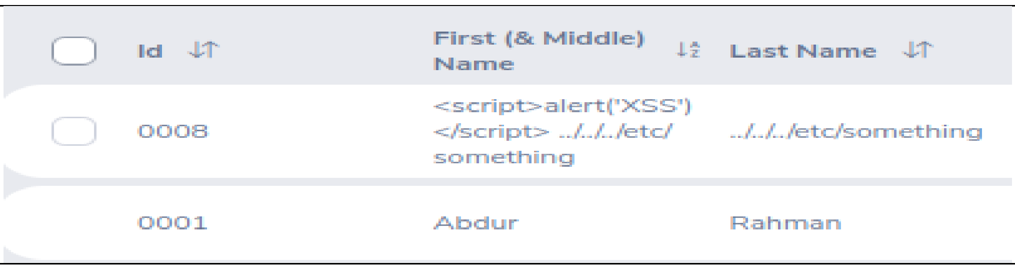
<script>alert('XSS')</script>

../../../etc/something

../../../etc/something

Employee Id

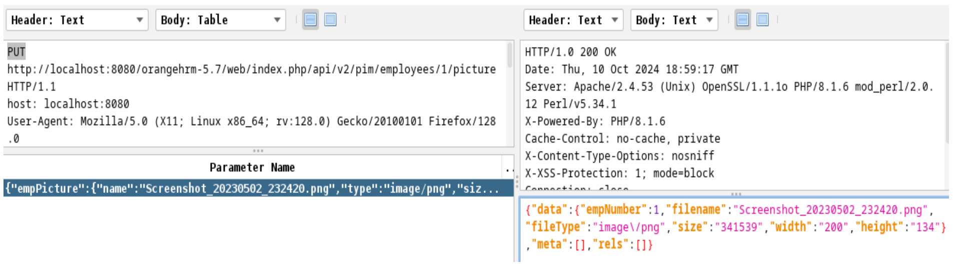
0008



Id	First (& Middle) Name	Last Name
0008	<script>alert('XSS')</script> ../../../etc/something	../../../etc/something
0001	Abdur	Rahman

Expectation	The application should have rejected the user creation attempt with inappropriate entries such as scripts, code, or any link.
Observation	It is observed that the system allows such entries in this module (though it was not allowed in the registration option).
Risk	Yes

TC04. To insert with improper file name to do Path Traversal Attacks

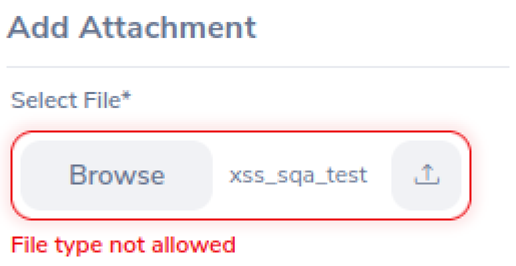
Goal	Target for this test case is to make a Path Traversal Attack by modifying uploaded file's name. If it is successful, then whenever a file is uploaded, it should redirect to another path (in this case to the profile menu, see 5th step) immediately.
Steps	● Start ZAP as a proxy. ● Go to My Info → Personal Details ● Click on profile icon ● Upload a valid image (jpg/png) ● Intercept the request in ZAP and modify the filename to "../../../empNumber/1" ● Check if it is updated 

```
PUT
http://localhost:8080/orangehrm-5.7/web/index.php/api/v2/pim/employees/1/picture
HTTP/1.1
host: localhost:8080
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:128.0) Gecko/20100101 Firefox/128
.0
{"empPicture":{"name":"Screenshot_20230502_232420.png","type":"image/png","siz...
```

```
HTTP/1.0 200 OK
Date: Thu, 10 Oct 2024 18:59:17 GMT
Server: Apache/2.4.53 (Unix) OpenSSL/1.1.1o PHP/8.1.6 mod_perl/2.0.
12 Perl/v5.34.1
X-Powered-By: PHP/8.1.6
Cache-Control: no-cache, private
X-Content-Type-Options: nosniff
X-XSS-Protection: 1; mode=block
{"data":{"empNumber":1,"filename":"Screenshot_20230502_232420.png",
"fileType":"image\/png","size":"341539","width":"200","height":"134"}
,"meta":[],"rels":[]}
```

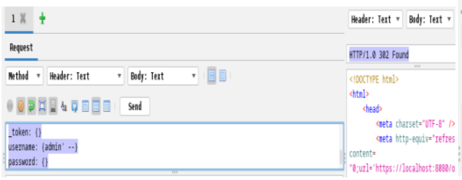
Expectation	File name should not be changed as file path fashion, rather should have sanitation mechanism so that it prevents from renaming them to inappropriate names.
Observation	File Name is not changed. Put request was unmodified
Risk	None

TC05. To upload vulnerable XSS payloads in the file attachment

Goal	Similar to TC04, this test was also designed for testing file upload criteria in the My Info module. Target for this test case is to upload a XSS script as executable file.
Steps	● Start ZAP as a proxy. ● Go to My Info → Personal Details ● Select Add Attachment option ● Upload an XSS file ● Check if it is uploaded  Add Attachment Select File* Browse xss_sqa_test File type not allowed
Expectation	Executable files should not be allowed to be uploaded
Observation	Executable files are not allowed.
Risk	None

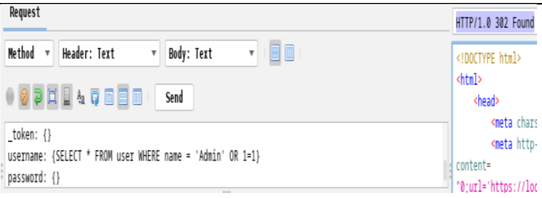
TC06. To check SQL injection vulnerability in the login functionalities

Goal	Goal for this TC was to verify if the application is secure against SQL injection attacks during login.
Steps	● Start ZAP proxy browser (top right) ● Go to the login page ● Try a login with any username/password (as we need the request header) ● Intercept request, and get the parameter name ● Modify the request in *Repeater* ● Put different SQL injection queries ● Tested ones: *admin'* --; *admin' OR '1'=';, SELECT * FROM user WHERE name = 'Admin' OR 1=1;* ● Keep the other entries empty in the request Repeater



```
Request
Method  Header: Text  Body: Text
_token: {}
username: {admin' --}
password: {}
```

```
HTTP/1.0 302 Found
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8" />
        <meta http-equiv="refres
content=
"0;url='https://localhost:8080/o
```



```
Request
Method  Header: Text  Body: Text
_token: {}
username: {SELECT * FROM user WHERE name = 'Admin' OR 1=1}
password: {}
```

```
HTTP/1.0 302 Found
<!DOCTYPE html>
<html>
    <head>
        <meta chars
        <meta http-
content=
"0;url='https://loc
```

Expectation	SQL injections should not be allowed
Observation	SQL injections are not allowed in the login module.
Risk	None

TC07. To check for bruteforce attempts vulnerability in the login functionalities

Goal	In this test case, vulnerability against bruteforce attack was evaluated. It was also done on login module. The plan was that: first a known username has to be present, then using that name, multiple attempts on password will be carried out. Although it is obvious that it will be nearly impossible to hit a correct pswd (as pswd is 8=< chars long with upper/lower case, and special character constrains), so what actually was check is the response from the site after sending multiple attempts from the same username.
Steps	● Start ZAP proxy browser (top right) ● Go to the login page ● Put valid username in the username field (but keep password black) ● Get a login request in ZAP ● Open Intruder → Positions → Payloads ● Add 8-digit passwords in the Payloads tab (or import in a txt file) ● Select attack type as Sniper (to avoid ram issue)
Expectation	Multiple attempts from same username should provide different response (like “try again”, or “account locked”, or “contact admin”, or something of that sorts.
Observation	No response from the site was noted even after multiple (10 actually) attempts. The site seemed to allow these attempts without flagging any warning.
Risk	Yes

TC08. To check the access control to the users for *candidates'* application

Goal	For this test case, it was check whether access control in in the *Recruitment* module can be altered. The Recruitment module contains Candidates and Vacancies options where hiring manager (could be any user) makes post mentioning the job title, description, etc as vacancy posts. On the other hand, the candidates makes application to those posts. When done, the hiring manager can short-list/reject the applicant. Now target for this test case is to see if it is possible to non-hiring managers (i.e., normal user/candidates) can alter this application system (e.g. reject it, or short-list it) themselves.
Steps	● Login as hiring manager ● Make vacancy post ● Login as normal user (i.e., as a candidate) ● Apply for the post ● Log out ● Get the url of hiring manager ('s Recruitment module) ● Try to short-list/reject the applications
Expectation	Applications should be short-listed/rejected only by the respective hiring managers
Observation	Only the hiring manager and the admin can short-list/reject the applicants.
Risk	None (however the admin should not be allowed to short-list/reject the applicants); Besides no log file was generated so it cannot be said if it is the admin or the hiring manager who short-list/reject the applicants

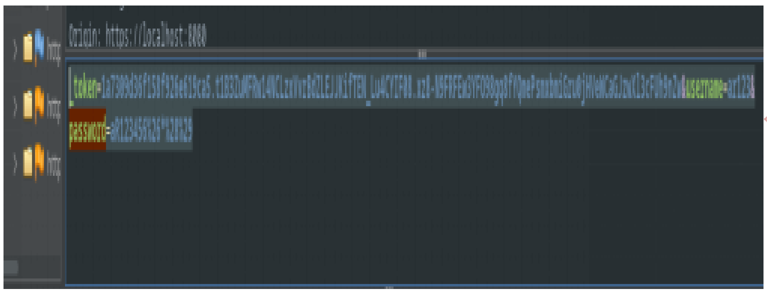
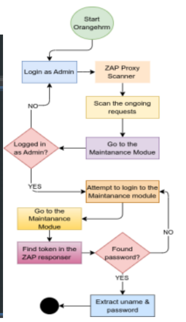
TC09. To perform the purge action to all users and vacancy posts

Goal	In the Maintenance module, there is a 2nd login form available for the admin access. In this module, the admin can see all the purged user's details, as well as purge users and vacancy posts. So, for this test case, it was tested whether a non-admin can do it using MITM attack. Ettercap was used to carry out this attack using its ARP poisoning module. In the console, it should show the input text fields' (username, and password) entries. So the target is to get the entries and try to login as admin to the Maintenance module and try to purge all the information of users/vacancy posts.
Steps	● Login as admin (must) ● Start Ettercap as root user ● Scan for the hosts ● Select router's IP (gateway) as target 1 ● Select admin's IP as target 2 ● Select ARP poisoning option (top-right)

	● Start MITM ● Look for input fields in the Ettercap's console
Expectation	The credential information should not be visible in the Ettercap's console
Observation	No info was noted
Risk	None

TC10. To perform the purge action to all users and vacancy posts

Goal	The goal for this TC is similar to TC09. However, in this TC, instead of Ettercap, ZAP was used.

Steps	● Login as admin (must)
	● Start ZAP's Spider
	● Scan for all the available ports (and respective requests)
	● Find "auth" request
	● Look for responses in the field.
	● Get the token, uname, and pswd
	● Make PUT request and add them in it.
	
	`Origin: https://localhost:8080`
	`token=[illegible]&username=ar1234`
	`password=ar123456%26*%28%29`
	
	 Start Orangehrm
	Login as Admin
	ZAP Proxy Scanner
	Scan the ongoing requests
	Go to the Maintanance Modue
	Logged in as Admin?
	NO
	YES
	Attempt to login to the Maintanance module
	Go to the Maintanance Modue
	Find token in the ZAP responser
	Found password?
	NO
	YES
	Extract uname & password

Expectation	The credential information should not be visible in the response; even if it should, then it it should have been encrypted.
Observation	All username, token, and password were found in plain text (w/o any encryption or compression.
Risk	Yes

TC11. To validate input sanitization in user registration forms

Goal	The goal for this TC is to ensure whether the user registration forms correctly sanitize inputs to prevent injection attacks and ensure that only valid data is accepted in the registry form.
Steps	● Go to installation step of orangehrm (as the site doesn't offer new account creation in the login page) ● Locate fields like username, password, email, and full name ● Test a list of various malicious entries for checking sanitization ● Tested ones (in each field (as entries): Category / Attempts / Result: SQL / '; DROP TABLE users; -- / × XSS / <script>alert('SQA Test')</script> / × HTML / <a href="uiu.ac.bd">link test</a> / × Characters / !@#$%^&*() / ×
Expectation	Malicious entries should not be allowed as executing (or reading) them will cause security issue.
Observation	Malicious entries in the registration are not allowed. In fact, the system required a username of minimum 5 chars, and password of 8<= chars with strict conditions.
Risk	None

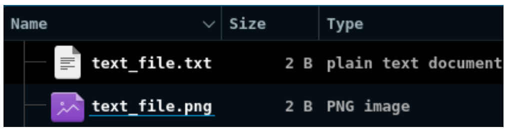
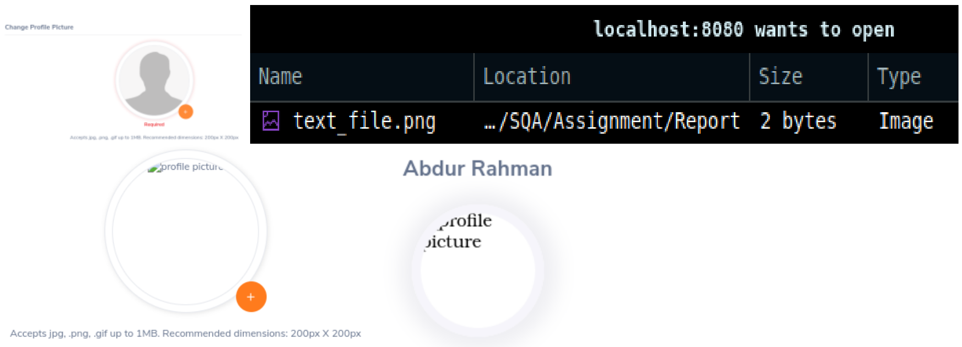
Category	Attempts	Result
SQL	'; DROP TABLE users; --	×
XSS	<script>alert('SQA Test')</script>	×
HTML	<a href="uiu.ac.bd">link test</a>	×
Characters	!@#$%^&*()	×

TC12. To test for session expiration after logout

Goal	The goal for this TC is to check if the user sessions are properly invalidated upon logout preventing unauthorized access to sensitive areas of the application after a user has logged out. If the sessions can be accessed again after expiration using that session, unauthorized access to the site can be made.
Steps	● Login with the proxy browser of ZAP ● Monitor the requests (ensure login action is captured) ● Go to (any) other section (for instance, viewEmployeeList), and establish an active session ● Capture that request in ZAP as well ● Logout ● Inspect the session tokens used before and after the logout request ● Use ZAP's "Request Editor" to manually enter URLs for restricted areas (e.g., http://localhost:8080/orangehrm-5.7/web/index.php/pim/viewEmployeeList). ● Send the request and monitor the response.

Expectation	Once the session is expired, it should redirect to login page (not allow to use that same session)
Observation	It redirects to the login page rather than allowing to access the PIM module w/o new session.
Risk	None

TC13. To verify that file uploads are restricted to allowed types only

Goal	This is somewhat similar to TC04 and TC05. However, instead of trying to upload XSS payload or File Traversal Attack, in this TC the file-upload criteria is validated. It is tested whether unacceptable files can be uploaded (encoding it to acceptable types). For instance, if the site supports jpg/png, and in that attachment other files (scripts/pdf) can be uploaded by changing their extensions.
Steps	● Login → My Info → Personal Details ● Select profile icon ● Upload a file (which is actually pdf/txt, but renamed to .jpg extension) ● Click upload ● Select “Add” in the Attachment tab ● Upload a script ● Click “Save” Both of the files below were of txt type, but the 2nd one was *renamed* to png extension.  Name | Size | Type text_file.txt | 2 B | plain text document text_file.png | 2 B | PNG image  localhost:8080 wants to open Name | Location | Size | Type text_file.png | …/SQA/Assignment/Report | 2 bytes | Image Change Profile Picture Required Accepts jpg, .png, .gif up to 1MB. Recommended dimensions: 200px X 200px profile picture Accepts jpg, .png, .gif up to 1MB. Recommended dimensions: 200px X 200px Abdur Rahman profile picture
Expectation	Along with file extension, the content should also be validated. If the file type if not of jpg/png (for images), then it should not be allowed to be uploaded regardless of it “renamed” extension

Observation	For the profile picture, it was noticed that file (of not picture type) can be uploaded in the picture section if it has jpg/png extension. For the attachment part (below), it accepted pdf files (within 1MB), but any executable was denied uploading.
Risk	Yes (along with extension, file content should've been verified)

TC14. To scan for vulnerabilities in the Buzz Module

Goal	The goal for this TC is to automatically detect vulnerabilities in the Buzz Newsfeed posting feature using OWASP ZAP's active scanner. The main focus is to check if any inappropriate content (e.g., scripts, sql injection, command queries, direct link, etc.) can be posted in the Buzz Newsfeed feature.
Steps	● Login is with the proxy browser of ZAP ● Go to the Buzz module ● Write post (include scripts, sql injection, command queries, direct link, etc.) ● Click post ● In ZAP, locate the request for posting content under the "Sites" tab. ● Select "Attack" → "Active Scan". ● When done, navigate to the "Alerts" tab ● Check the Alerts (Low/Medium/High)
Expectation	The content of the post should be sanitized to filter vulnerable contents
Observation	No filtering observed; all posts are as they are made. Besides ZAP's scanner did not flag anything as "risky"
Risk	Undecided

TC15. To test SQL injection in the *search* functionalities

Goal	The goal for this TC is to identify potential SQL injection vulnerabilities in the search functionalities of the application.
Steps	● Login with the proxy browser of ZAP. ● Go to the Search bar. ● Perform a search with normal (w/o sql queries) entries. ● Locate the search request under the "Sites" tab. ● Right-click on the request and select "Send to Repeater." ● Open the Repeater tab and modify the search input with common SQL

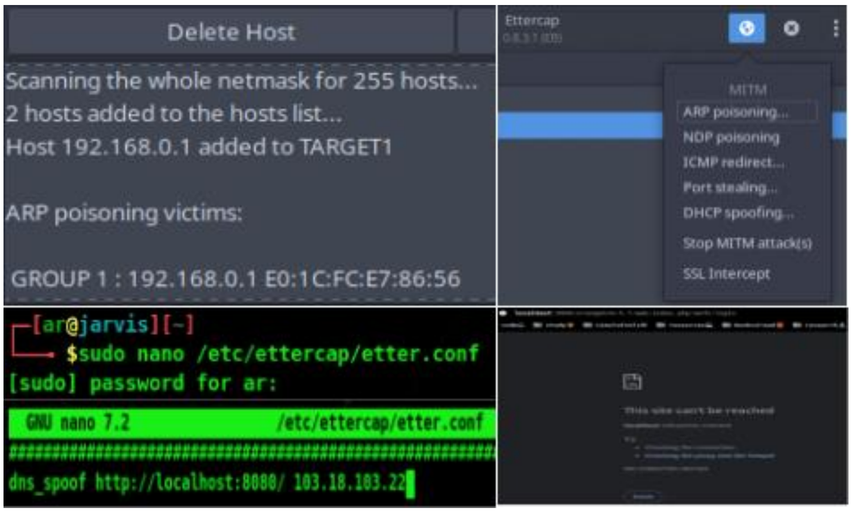
	injection payloads, such as: ➢ ' OR '1'='1 ' ➢ UNION SELECT NULL, username, password FROM users -- ➢ '; DROP TABLE users; -- ● Return to the “Sites” tab → select “Attack” > “Active Scan.” ● When done, navigate to the “Alerts” tab. ● Look for any alerts related to SQL injection
Expectation	Such search entries should not be able to modify any data inside the site.
Observation	No alteration in the data is noted
Risk	None

TC16. To check for CSRF protection on form submission

Goal	The goal for this TC is to verify that the application implements Cross-Site Request Forgery (CSRF) protection on sensitive form submissions.
Steps	● Login with the proxy browser of ZAP. ● Access any form (say PIM’s Employee List) ● Perform a submission with normal entries. ● Capture the request in ZAP → Go to “Sites” tab ● Manually check for any CSRF tokens present in the request (look for hidden fields or headers) ● Go to the “Tools” → select “Anti-CSRF Tokens.” → Enable ● Return to the “Sites” tab → select “Attack” > “Active Scan.” ● When done, navigate to the “Alerts” tab. ● Look for any identified CSRF vulnerabilities.
Expectation	If the CSRF token is invalid or missing, the application should reject the request or provide alert or notification of that sorts.
Observation	No notification was seen
Risk	None

TC17. To check DNS spoofing for redirecting traffics

Goal	Goal for this TC is to test whether DNS spoofing can redirect traffic within the site to a malicious site.

Steps	● Launch Ettercap (as root)
	● Go to “Sniff” → Unified Sniffing → select “network interface”
	● Scan for hosts → Host → Host List
	● Set router’s gateway (192.168.0.1) as Target 1
	● Set http://localhost:8080/ as Target 2
	● Go to Plugins → Manage Plugins → dns_spoof
	● Go to the MITM tab→ ARP poisoning.
	● Start → Start Sniffing
	● Check if the request is redirected to another IP (gateway).
	
	Delete Host
	Scanning the whole netmask for 255 hosts...
	2 hosts added to the hosts list...
	Host 192.168.0.1 added to TARGET1
	ARP poisoning victims:
	GROUP 1 : 192.168.0.1 E0:1C:FC:E7:86:56
	`┌─[ar@jarvis][~]`
	`└──╼ $sudo nano /etc/ettercap/etter.conf`
	`[sudo] password for ar:`
	`GNU nano 7.2 /etc/ettercap/etter.conf`
	`############################################`
	`dns_spoof http://localhost:8080/ 103.18.103.22`
	This site can't be reached
	 Ettercap
	MITM
	ARP poisoning...
	NDP poisoning
	ICMP redirect...
	Port stealing...
	DHCP spoofing...
	Stop MITM attack(s)
	SSL Intercept
	(malicious IP list: https://www.projecthoneypot.org/list_of_ips.php)

Expectation	If attack such as this is properly addressed, the site still can be visited in the local host without any redirection.
Observation	The site was not not visited, neither was the redirected site (IP).
Risk	Undecided (this could be an issue due to the fact that the site was running on the local host (which do not have secure transfer protocol (i.e, https)

5. Conclusion

Throughout the tests, it was checked whether orangehrm was secure against some of the common attacks. The test cases included: Unauthorized Access and Privilege Escalation (TC01. TC02, TC08), Input Validation and Injection Attacks (TC03, TC06, TC11, TC15), File Upload Vulnerabilities (TC04, TC05, TC13), Brute Force and Session Management (TC07, TC12), Cross-Site Request Forgery (TC16). Data Purging (TC09, TC10), General Vulnerability Scanning (TC14), and Man-in-the-Middle attacks (TC17). it is worth noting that all the test cases were conducted on the local host, thus the tests on live server might provide different results.