Secure Coding WS 2014
Phase 4: White-Box Testing

Team 6

Shady Botros
Subburam Rajaram
Vasudhara Venkatesh

Executive Summary

This report contains the results of the White-box testing of team 7's web application, "NoName". The testing process was carried-out systematically by following the guidelines of the Open Web Application Security Project (OWASP) White box testing checklist provided. In this report, we do not only mention the vulnerabilities of the application and their impact, but also give recommendations on how to fix them.

.

Table of Contents

Time-Tracking Table

Student	Task	# of Hours
Subbu	Learn tools and attacks	6
	Team 7 Code review	4
	White box and functionality testing	7
	Black box testing using nmap, w3af and sqlmap	4
	Tried Reverse Engineering the C code using IDA	2
	Presentation, WB testing Report and Checklist	5
		28 hours
Shady	Background reading: OWASP Testing Guide	2
	Explore web app and source code review	5
	BB Testing using ZAP and manually	3
	WB testing using RIPS and eliminating false positives by reviewing source code	9
	Design XSS and CSRF attack and recorded video for the demo	2
	Presentation, testing report and checklist	6
		27 hours
Vasudhara Venkatesh	Study OWASP Testing Guide	5

	Identity Management Testing,Authentication Testing	5
	Session Management Testing	7
	Rips tool Exploration	8
	Used Black box testing tools to verify	4
	Testing Report Consolidation	4
		33 hours

Team 7's App Vulnerabilities

1. Configuration and Deploy Management Testing

1.1 Application Platform Configuration

Status : Not Tested

Description: During the White box testing, we couldn't get access to the Virtual Machine. So Application Platform configuration is not tested.

1.2 Test File Extensions Handling for Sensitive Information

Likelihood	Medium
Impact	Medium
Risk	Medium

Observation: During the White box testing, we identified that Team 7's app is not testing for the file extensions when user is performing for the Batch Transaction. This leads to uploading the executable file extensions like .php, .out files into web server.

Discovery: While uploading the file for the batch transactions the file extension was not checked in the app. This is was discovered during testing for functionality.

Likelihood: This requires the user to access the transfer functionality and knowledge about the linux and php commands to exploit the system running the app.

Impact: This will be huge exploit if the execution of file takes place on the system level or /var/www level user.

Recommendation: File extension and validation for the extra unnecessary input should be validated.

1.3 Backup and Unreferenced Files for Sensitive Information

Likelihood	Medium
Impact	Low
Risk	Low

Observation: During the White box testing, we identified that Team 7's app uses unreferenced files which contain sensitive information

Discovery: It was identified that there are .git files present which are accessible over browser. This contains the source code of the application and other data.

This vulnerability can be identified by using dirbuster or pykto tools.

Likelihood: This requires the user to access URLs not served by application and use of tools such as dirbuster , pykto etc.

Impact: Directory listing has not been disabled in the server and many files not served by the applications could be indexed which could reveal information which was not intended.

Recommendation: Use of default server configuration should be avoided. Additional security configuration to disallow files should be configured.

1.4 Test Http Methods

Status: Does not apply

Description: During the White box testing, we identified that Team 7's app uses Https protocol and port 80 for http is closed down.

Discovery: It was identified by running the command

telnet localhost 80
OPTIONS / HTTP/1.1

1.5 Test HTTP Strict Transport Security

Likelihood	Medium
Impact	Medium
Risk	Medium

Observation: During the White box testing, we identified that Team 7's application does not use HTTP strict transport security.

Discovery: This can be identified by running the below command. It shows that HSTS header are not enabled.

```
curl -s -D- https://127.0.0.1/ | grep Strict
```

Likelihood: The vulnerability could be exploited through a man in the middle attack.

Impact: If exploited , the data sent to the server could be tampered by a third party

Recommendation: The HSTS header should be enabled.
HSTS can be enabled in Apache with mod headers and the following line in the configuration:

```
<IfModule mod_headers.c>
 # this domain should only be contacted in HTTPS for the next 6
months
 Header add Strict-Transport-Security "max-age=15768000"
</IfModule>
```

1.6 Test RIA cross domain policy

Status : Secure

Observation: During the White box testing, we tried to access the policy files crossdomain.xml and clientaccesspolicy.xml but these files are not being used in Team 7's app.

Discovery: Tried accessing the below URLs and they were not successful. Additionally the w3af tool did not report these vulnerabilities.
http://localhost.com/crossdomain.xml and

http://localhost.com/clientaccesspolicy.xml.

Likelihood: This requires the user to be aware of the vulnerability of the policy files and so it can be classified as low likelihood.

Impact: It defeats CSRF protections.

Recommendation: Use of default server configuration should be avoided. Additional security configuration is to be used.

2. Identity Management Testing

2.1 Test Role Definitions

Status : Secure

Observation: During the White box testing, we identified that Team 7's application is not vulnerable to Identity related attacks.

Discovery: By accessing the application using different user roles the below role matrix was developed and tested for conformity.

ROLE	PERMISSION	OBJECT	CONSTRAINTS
Admin	Read	Employee records, Customer records	Records associated with Employees and Customers
Employee	Read	Customer records	Only records associated with customers
Client	Read	Customer record	Only own record

Additionally below features were noticed to be present in the application

- Client has to be approved by admin or employee
- Only the Employee or Admin can only provision users
- Invalid credentials are returned with error
- Account names are client email addresses and they are random

Likelihood: This is less likely in applications as its part of functionality.

Impact: The application should have proper role definitions, failing which will cause privilege escalation attacks.

Recommendation: Proper role definitions would need to be established in the application and tested.

3. Authentication Testing

3.1 Testing for Credentials Transported over an Encrypted Channel

Status : Secure

Observation: During the White box testing, we saw that connection between the client and server is taking place using https encrypted connection.

Discovery: We used debugger tools of firefox to see the Https Post and Get Connections.

Likelihood: This requires the user to be aware of the Https and Https protocol and debugger tool from which page we came from.

Impact: Potential leakage of the user credentials for the attackers.

Recommendation: Use Https connection for both POST and GET.

3.2 Testing for default credentials

Status : Secure

Observation: During the White box testing, we tried the common password for admin and employees and for users listed. But no default credentials are used.

Discovery: We tried the normal functionality testing for the entering credentials listed by owasp testing guide.

Likelihood: This requires the user to be aware of some of the default credentials used.

Impact: Potential leakage of the user credentials for the attackers.

Recommendation: Use strong passwords with combinations of characters.

3.3 Testing for weak lock out mechanism

Likelihood	Medium
Impact	Medium
Risk	Medium

Observation: During the White box testing, we saw that there is no lock out mechanism for the credentials being tried continuously i.e brute force attack.

Discovery: We used ‘n’ times and review the code for the lockout mechanism and unlock mechanism.

Likelihood: The attackers must poses potential automated system to brute force attacks.

Impact: Potential leakage of the user credentials for the attackers.

Recommendation: Use potential lockout mechanism after 3 or 5 trails and safe unlock mechanisms to unlock user.

4 Testing for Session Management Schema

4.1 Testing for Bypassing Session Management Schema

Likelihood	Medium
Impact	High
Risk	High

Observation: During the White box testing, we saw that there is no secure flag for the cookie set. This will lead to the session hijack.

Discovery: We used developer tools firefox to see the cookie flag is set to secure or not.

Likelihood: The attackers knowledge of session ids, cookies, randomness of the previous things generations will lead for potential session hijack.

Impact: Potential leakage of the user session for the attackers.

Recommendation: Use secure flag for the cookie and strong cookie generation algorithm.

4.2 Testing for Cookies attributes

Likelihood	Medium
Impact	High
Risk	High

Observation: During the White box testing, we saw that the cookie path attribute has been set to root path “/”. if the path is set to the root directory "/" then it can be vulnerable to less secure applications on the same server. For example, if the application resides at /myapp/, then verify that the cookies path is set to "; path=/myapp/" and NOT "; path=/" or "; path=/myapp". Notice here that the trailing "/" must be used after myapp. If it is not used, the browser will send the cookie to any path that matches "myapp" such as "myapp-exploited".
Discovery: We used developer tools firefox to see the cookie attributes are correctly from guidelines of the owasp.

Likelihood: The attackers knowledge of session ids, cookies will lead for potential attack on the other apps on the server.

Impact: Potential access to the other apps on the server.

Recommendation: Correctly set the cookie attributes using the owasp guidelines..

4.3 Testing for Session Fixation

Status : Not Tested

Description: When an application does not renew its session cookie(s) after a successful user authentication, it could be possible to find a session fixation vulnerability and force a

user to utilize a cookie known by the attacker. In that case, an attacker could steal the user session (session hijacking).

Likelihood: The attackers knowledge of session ids, cookies will lead for potential session hijack.

Impact: Potential access to the Session of user by attacker.

Recommendation: Developers should have implemented a session token renew after a user successful authentication.

4.4 Testing for Exposed Session Variables

Status : Not Tested

Description: The Session Tokens (Cookie, SessionID, Hidden Field), if exposed, will usually enable an attacker to impersonate a victim and access the application illegitimately. It is important that they are protected from eavesdropping at all times, particularly whilst in transit between the client browser and the application servers.

Likelihood: The attackers knowledge of session ids, cookies will lead for potential stealing of the data.

Impact: The attackers knowledge of session ids, cookies will lead for potential stealing of the data.

Recommendation: Potential session variables should be sent over encrypted connection.

4.5 Testing for Cross Site Request Forgery

Likelihood	Medium
Impact	High
Risk	High

Description: CSRF is an attack which forces an end user to execute unwanted actions on a web application in which he/she is currently authenticated. With a little help of social engineering (like sending a link via email or chat), an attacker may force the users of a web application to execute actions of the attacker's choosing. A successful CSRF exploit can

compromise end user data and operation, when it targets a normal user. If the targeted end user is the administrator account, a CSRF attack can compromise the entire web application.

Likelihood: The attackers knowledge of CSRF token and its utilization by using the script to get the data is required.

Impact: The attackers knowledge CSRF will potentially leak the data of the users to the attacker.

Recommendation: Potential session variables should be sent over encrypted connection.

4.6 Testing for logout functionality and

Status : Not Tested

Description: Session termination is an important part of the session lifecycle. Reducing to a minimum the lifetime of the session tokens decreases the likelihood of a successful session hijacking attack. This can be seen as a control against preventing other attacks like Cross Site Scripting and Cross Site Request Forgery.

Likelihood: The attackers knowledge of session ids, session termination will potentially lead to re login to app.

Impact: The attackers knowledge of session ids t and session termination lead to potential expolit of the users data.

4.7 Test Session Timeout

Status : Not Tested

Description: Session termination is an important part of the session lifecycle. Reducing to a minimum the lifetime of the session tokens decreases the likelihood of a successful session hijacking attack. This can be seen as a control against preventing other attacks like Cross Site Scripting and Cross Site Request Forgery.

Likelihood: The attackers knowledge of session ids, session termination will potentially lead to re login to app.

Impact: The attackers knowledge of session ids, *Session Timeouts* and session termination lead to potential expolit of the users data.

4.8 Testing for Session puzzling

Status : Not Tested

Description: Session Variable Overloading (also known as Session Puzzling) is an application level vulnerability which can enable an attacker to perform a variety of malicious actions, including by not limited to:

- Bypass efficient authentication enforcement mechanisms, and impersonate legitimate users.
- Elevate the privileges of a malicious user account, in an environment that would otherwise be considered foolproof.
- Skip over qualifying phases in multi-phase processes, even if the process includes all the commonly recommended code level restrictions.
- Manipulate server-side values in indirect methods that cannot be predicted or detected.
- Execute traditional attacks in locations that were previously unreachable, or even considered secure.

Likelihood: An authentication bypass attack vector could be executed by accessing a publicly accessible entry point (e.g. a password recovery page) that populates the session with an identical session variable, based on fixed values or on user originating input.

Impact: The attackers knowledge of session ids, Session Timeouts and session variables lead to potential expolit of the users data.

Our vulnerabilities

1. Configuration and Deploy Management Testing

1.1 Application Platform Configuration - Vulnerable

Likelihood	Medium
Impact	Medium
Risk	Medium

Observation: During the White box testing, we identified that our application uses the default configuration for the web server.

Discovery: It was identified that its possible to perform directory traversal. Use of w3af confirmed the vulnerability

Likelihood: This requires the user to access URLs not served by application and use of tools such as dirbuster , pykto etc.

Impact: Directory listing has not been disabled in the server and many files not served by the
applications could be indexed which could reveal information which was not
intended.

Recommendation: Use of default server configuration should be avoided. Additional security configuration is to be used.

1.2 File Extensions Handling for Sensitive Information - Vulnerable

Likelihood	Low
Impact	Medium
Risk	Medium

Observation: During the White box testing, we identified that our application uses well known File Extensions Handling for Sensitive Information

Discovery: It was identified that a well known file config.php had the credentials to access the DB

This vulnerability can be identified by using dirbuster or pykto tools.

Likelihood: This requires the user to access the file with sensitive information. However this is not directly served by the application.
So this vulnerability has a low likelihood

Impact: If the files with sensitive information is identified, then it could be used for further exploits.

Recommendation: The application should not use well known filenames and file extensions for files which have sensitive information

1.3. Old, Backup and Unreferenced Files for Sensitive Information - Vulnerable

Likelihood	Low
Impact	Medium
Risk	Medium

Observation: During the White box testing, we identified that our application uses unreferenced files which contain sensitive information

Discovery: It was identified that there are .git files present which are accessible over browser. This contains the source code of the application and other data.

This vulnerability can be identified by using dirbuster or pykto tools.

Likelihood: This requires the user to access URLs not served by application and use of tools such as dirbuster , pykto etc.

Impact: Directory listing has not been disabled in the server and many files not served by the
applications could be indexed which could reveal information which was not intended.

Recommendation: Use of default server configuration should be avoided. Additional security configuration to disallow files should be configured.

1.4 HTTP Strict Transport Security - Vulnerable

Likelihood	Medium
Impact	Medium
Risk	Medium

Observation: During the White box testing, we identified that our application does not use HTTP strict transport security.

Discovery: This can be identified by running the below command. It shows that HSTS header are not enabled.

```
curl -s -D- https://localhost.com/ | grep Strict
```

Likelihood: The vulnerability could be exploited through a man in the middle attack.

Impact: If exploited , the data sent to the server could be tampered by a third party

Recommendation: The HSTS header should be enabled.
HSTS can be enabled in Apache with mod headers and the following line in the configuration:

```
<IfModule mod_headers.c>
 # this domain should only be contacted in HTTPS for
months
 Header add Strict-Transport-Security "max-age=15768
</IfModule>
```

1.5 RIA cross domain policy - Not Vulnerable

Likelihood	Low
Impact	Low
Risk	Low

Observation: During the White box testing, we tried to access the policy files crossdomain.xml and clientaccesspolicy.xml but these files are not being used in our app.

Discovery: Tried accessing the below URLs and they were not successful. Additionally the w3af tool did not report these vulnerabilities. http://localhost.com/crossdomain.xml and http://localhost.com/clientaccesspolicy.xml.

Likelihood: This requires the user to be aware of the vulnerability of the policy files and so it can be classified as low likelihood.

Impact: It defeats CSRF protections.

Recommendation: Use of default server configuration should be avoided. Additional security configuration is to be used.

2. Identity Management Testing - Not Vulnerable

Likelihood	Medium
Impact	High
Risk	High

Observation: During the White box testing, we identified that our application is not vulnerable to Identity related attacks.

Discovery: By accessing the application using different user roles the below role matrix was developed and tested for confirmity.

ROLE	PERMISSION	OBJECT	CONSTRAINTS
Admin	Read	Employee records, Customer records	Records associated with Employees and Customers
Employee	Read	Customer records	Only records associated with customers
Client	Read	Customer record	Only own record

Additionally below features were noticed to be present in the application

- Client has to be approved by admin or employee
- Only the Employee or Admin can only provision users
- Invalid credentials are returned with error
- Account names are client email addresses and they are random

Likelihood: This is less likely in applications as its part of functionality.

Impact: The application should have proper role definitions, failing which will cause privilege escalation attacks.

Recommendation: Proper role definitions would need to be established in the application and tested.

3. Authentication Testing

3.1 Credentials Transported over an Encrypted Channel -Vulnerable

Likelihood	Medium
Impact	Medium
Risk	Medium

Observation: During the White box testing, we identified that although our application uses https, it is still possible to access the application folder using http.

Discovery: Secure content can be accessed using the insecure protocol HTTP. "http://target-host/CodingSecureWS2014/phase3/index.php"
Use of w3af testing tool confirmed this vulnerability.

Likelihood: Many websites use insecure HTTP protocol which can be exploited with Man in the middle attack.

Impact: If exploited , the data sent to the server could be tampered by a third party

Recommendation: Disallow insecure HTTP protocol and use of HTTPS.

3.2 Bypassing Authentication Schema - Not Vulnerable

Likelihood	Low
Impact	Medium
Risk	Medium

Observation: During the White box testing, we identified that our application is not vulnerable to bypassing authentication schema

Discovery: Examining the code, we identified that it is not vulnerable to bypassing authentication mechanism

Likelihood: The likelihood of this vulnerability could be low

Impact: Attacker could login without entering the password.

Recommendation: Ensure that the entered password is validated and is not susceptible to bypassing authentication or through Session ID prediction.

3.3 Weak password change or reset functionalities - Vulnerable

Likelihood	Low
Impact	Low
Risk	Low

Observation: During the White box testing, we identified that our application does not use any security mechanism for password reset

Discovery: On testing the app, the user would be able to access forgot password link and enter the email address of any valid user causing the password to be reset.

Likelihood: This requires the attacker to know the client's account.

Impact: The user could reset the password of any user and if the has access to the client's email address he would be able to access the banl account as well

Recommendation: The password reset function should require users tc re-authenticate or use additional security mechanism.

4. Authorization Testing

4.1 Directory traversal/file include Vulnerability - Vulnerable

Likelihood	Low
Impact	Medium
Risk	Medium

Observation: During the testing, we identified that our application uses the default configuration for the web server and allows directory traversal.

Discovery: It was identified that its possible to perform directory traversal. Use of w3af confirmed the vulnerability

Likelihood: This requires the user to access URLs not served by application and use of tools such as dirbuster , pykto etc.

Impact: Directory listing has not been disabled in the server and many files not served by the
applications could be indexed which could reveal information which was not intended.

Recommendation: Use of default server configuration should be avoided. Additional security configuration such as disallow directory indexing should be used.

5. Data Validation Testing

5.1 Cross site scripting - Vulnerable

Likelihood	Medium
Impact	High
Risk	High

Observation: During the web application vulnerability testing, we discovered that our app is
vulnerable for cross-site scripting attacks.

Discovery: The request where XSS attack can be found is
POST https://localhost/CodingSecureWS2014/phase3/login.php

Likelihood: Cross-site scripting (XSS) is a class of vulnerabilities affecting web applications that can result in security controls implemented in browsers being circumvented. When a
user visits a page on a website, script code originating in the website domain can access and manipulate the DOM (document object model), a representation of the page and its properties in the browser.

Impact: The precise impact depends greatly since the application is in banking domain
sharing sensitive data.
• XSS is generally a threat to web applications which have authenticated users or are
otherwise security sensitive.

• Malicious code may be able to manipulate the content of the site, changing its
appearance and/or function for another user.
• This includes modifying the behavior of the web application (such as redirecting
forms, etc).
• The code may also be able to perform actions within the application without user
knowledge.
• Script code can also obtain and retransmit cookie values if they haven't been set
HttpOnly.

Recommendation: The developer must identify how the untrustworthy data is being output to the client without adequate filtering.
There are various language/platform specific techniques for filtering untrustworthy data.
General rules for preventing XSS can be found in the recommended OWASP XSS
Prevention Cheat Sheet (see references).
For more information, please check-out the references below:
Cross-Site Scripting (Wikipedia)
Cross-Site Scripting (OWASP)
XSS Prevention Cheat Sheet

5.2 SQL Injection - Not Vulnerable

Likelihood	Medium
Impact	High
Risk	High

Observation: During the testing, we identified that our application is not vulnerable to SQL Injection

Discovery: We tested our app using pen testing tools such as w3af and sqlmap and also manually to see if the application is vulnerable to SQLi attacks.
Prepared statements have been used in the code by which SQLi has been avoided.

Likelihood: Injection flaws are very prevalent in legacy codes and can be exploited if parameterized
statements are not being used. SQL Injections can be detected via Manual testing or via scanners/fuzzers.
The SQL Injection could either be sent as a GET request or POST request and they could be with Integer or string based parameters.

Impact: SQL Injection can be used to access sensitive information from the database and can be used to create, modify or delete the contents of the database.
This application has a severe SQL Injection vulnerability which could be used by an attacker to access the contents of the database

Recommendation: Parameterized statements should be used to differentiate the code from the form data.
The input needs to be validated and whitelisted to prevent malicious data to be sent to the application which could result in revealing sensitive Database information.

6. Session Management Testing

6.1 Testing for Bypassing Session Management Schema

Likelihood	Medium
Impact	High
Risk	High

Observation: During the White box testing, we saw that there is no secure flag for the cookie set. This will lead to the session hijack.

Discovery: We used developer tools in firefox to see the cookie flag is set to secure or not.

Likelihood: The attackers knowledge of session ids, cookies, randomness of the previous things generations will lead for potential session hijack.

Impact: Potential leakage of the user session for the attackers.

Recommendation: Use secure flag for the cookie and strong cookie generation algorithm.

6.2 Testing for Cookies attributes - Vulnerable

Likelihood	Medium
Impact	High
Risk	High

Observation: During the White box testing, we saw that the cookie path attribute has been set to root path “/”. if the path is set to the root directory "/" then it can be vulnerable to less secure applications on the same server. For example, if the application resides at /myapp/, then verify that the cookies path is set to "; path=/myapp/" and NOT "; path=/" or "; path=/myapp". Notice here that the trailing "/" must be used after myapp. If it is not used, the browser will send the cookie to any path that matches "myapp" such as "myapp-exploited".
Discovery: We used developer tools firefox to see the cookie attributes are correctly from guidelines of the owasp.

Likelihood: The attackers knowledge of session ids, cookies will lead for potential attack on the other apps on the server.

Impact: Potential access to the other apps on the server.

Recommendation: Correctly set the cookie attributes using the owasp guidelines..

6.3 Testing for Session Fixation

Status : Vulnerable

Description: When an application does not renew its session cookie(s) after a successful user authentication, it could be possible to find a session fixation vulnerability and force a user to utilize a cookie known by the attacker. In that case, an attacker could steal the user session (session hijacking).

Likelihood: The attackers knowledge of session ids, cookies will lead for potential session hijack.

Impact: Potential access to the Session of user by attacker.

Recommendation: Developers should have implemented a session token renew after a user successful authentication.

6.4 Testing for Exposed Session Variables

Status : Not Tested

Likelihood	Medium
Impact	High
Risk	High

Description: The Session Tokens (Cookie, SessionID, Hidden Field), if exposed, will usually enable an attacker to impersonate a victim and access the application illegitimately. It is important that they are protected from eavesdropping at all times, particularly whilst in transit between the client browser and the application servers.

Likelihood: The attackers knowledge of session ids, cookies will lead for potential stealing of the data.

Impact: The attackers knowledge of session ids, cookies will lead for potential stealing of the data.

Recommendation: Potential session variables should be sent over encrypted connection.

6.5 Testing for Cross Site Request Forgery

Likelihood	Medium
Impact	High
Risk	High

Description: CSRF is an attack which forces an end user to execute unwanted actions on a web application in which he/she is currently authenticated. With a little help of social engineering (like sending a link via email or chat), an attacker may force the users of a web application to execute actions of the attacker's choosing. A successful CSRF exploit can compromise end user data and operation, when it targets a normal user. If the targeted end user is the administrator account, a CSRF attack can compromise the entire web application.

Likelihood: The attackers knowledge of CSRF token and its utilization by using the script to get the data is required.

Impact: The attackers knowledge CSRF will potentially leak the data of the users to the attacker.

Recommendation: Potential session variables should be sent over encrypted connection.

6.6 Testing for logout functionality

Status - Vulnerable

Likelihood	Low
Impact	Medium
Risk	Medium

Description: Session termination is an important part of the session lifecycle. Reducing to a minimum the lifetime of the session tokens decreases the likelihood of a successful session hijacking attack. This can be seen as a control against preventing other attacks like Cross Site Scripting and Cross Site Request Forgery.

Likelihood: The attackers knowledge of session ids, session termination will potentially lead to re login to app.

Impact: The attackers knowledge of session ids and session termination lead to potential expolit of the users data.

Recommendation: Make the logout functionality available and accessible to users at all time.

6.7 Test Session Timeout

Status :Vulnerable

Likelihood	Medium
Impact	Medium
Risk	Medium

Description: Session termination is an important part of the session lifecycle. Reducing to a minimum the lifetime of the session tokens decreases the likelihood of a successful session hijacking attack. This can be seen as a control against preventing other attacks like Cross Site Scripting and Cross Site Request Forgery.

Likelihood: The attackers knowledge of session ids, session termination will potentially lead to re login to app.

Impact: The attackers knowledge of session ids, *Session Timeouts* and session termination lead to potential expolit of the users data.

Recommendation: The attackers knowledge of session ids, Session Timeouts and session variables lead to potential expolit of the users data.

6.8 Testing for Session puzzling

Status : Not Tested

Description: Session Variable Overloading (also known as Session Puzzling) is an application level vulnerability which can enable an attacker to perform a variety of malicious actions, including by not limited to:

- Bypass efficient authentication enforcement mechanisms, and impersonate legitimate users.
- Elevate the privileges of a malicious user account, in an environment that would otherwise be considered foolproof.
- Skip over qualifying phases in multi-phase processes, even if the process includes all the commonly recommended code level restrictions.
- Manipulate server-side values in indirect methods that cannot be predicted or detected.
- Execute traditional attacks in locations that were previously unreachable, or even considered secure.

Likelihood: An authentication bypass attack vector could be executed by accessing a publicly accessible entry point (e.g. a password recovery page) that populates the session with an identical session variable, based on fixed values or on user originating input.

Impact: The attackers knowledge of session ids, Session Timeouts and session variables lead to potential expolit of the users data.