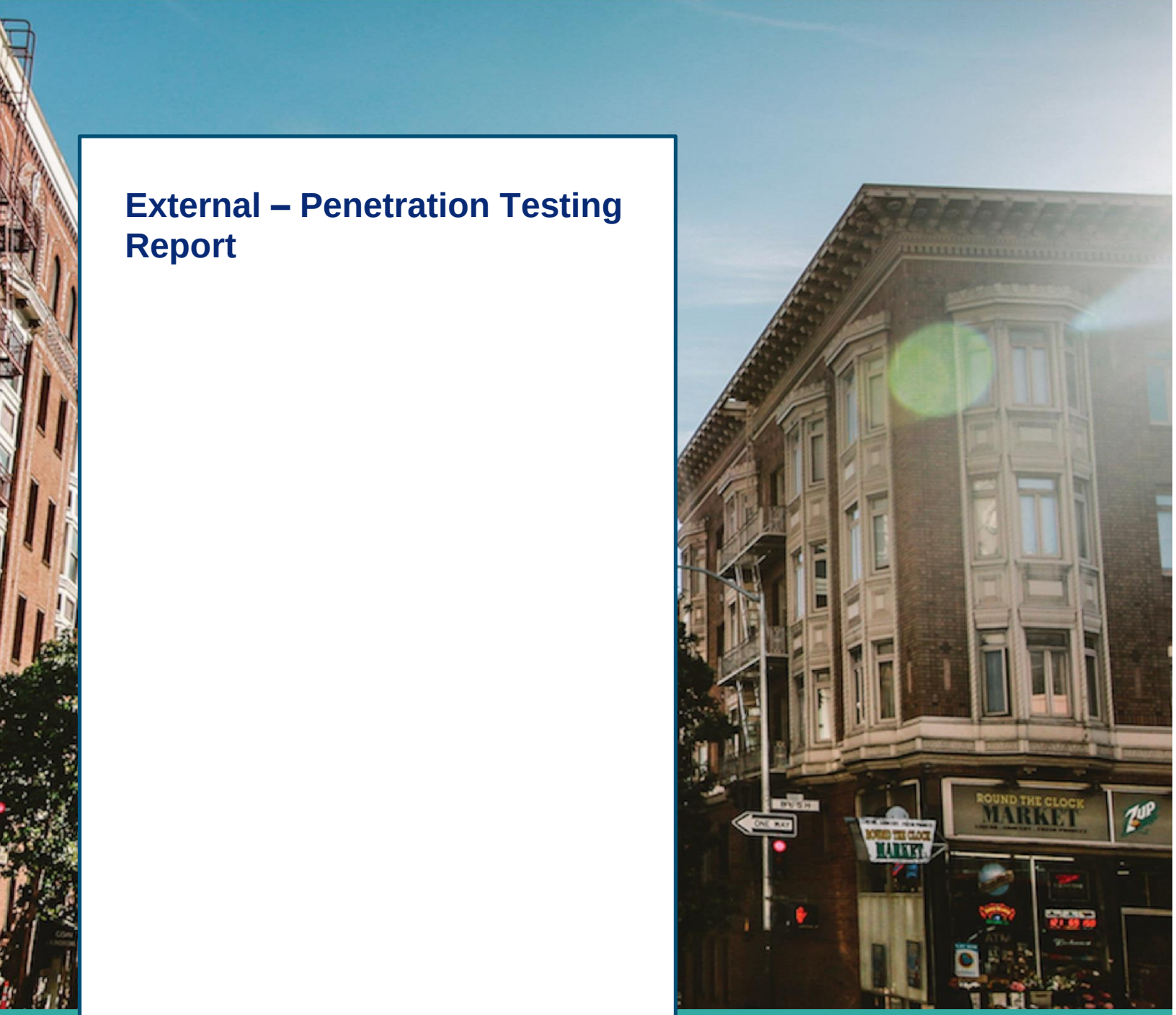
External – Penetration Testing Report



Confidential

Document Details

Document Title	External – Penetration Test Report
Document Version	1.0
Date	08-November-2019
Prepared By	Jeeva

Table of Contents

1. Executive Summary

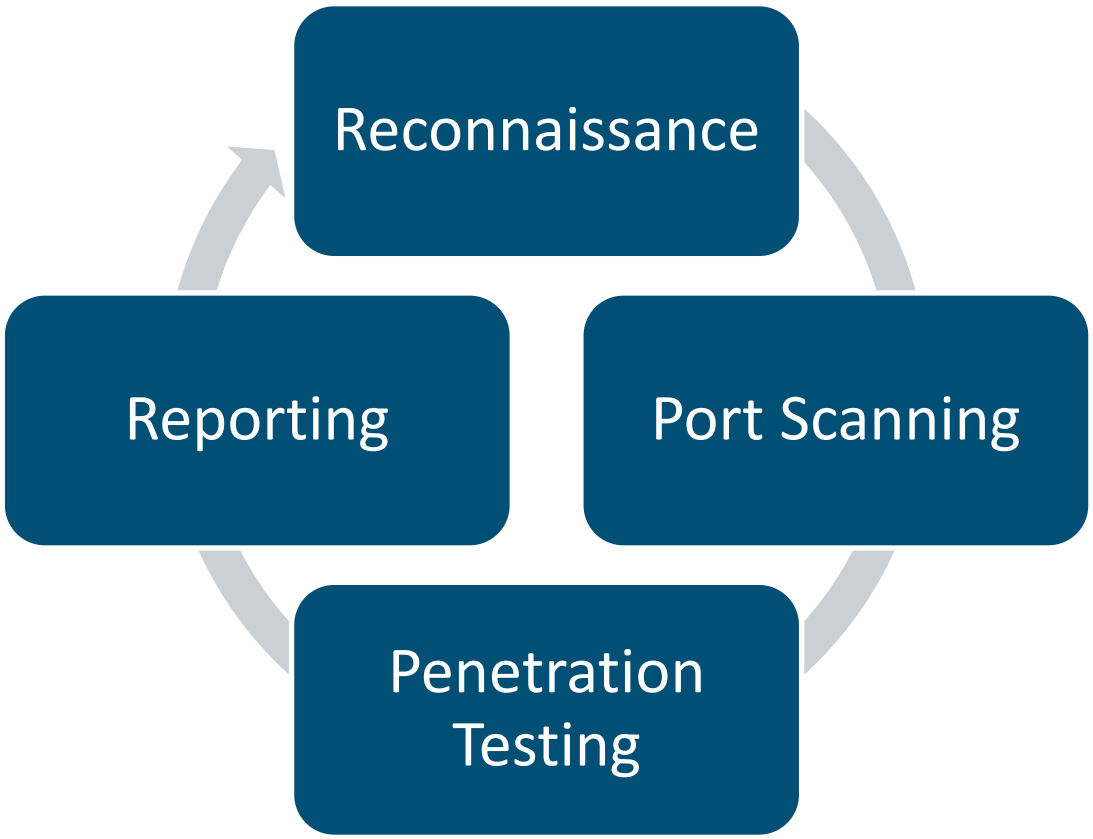
The goal was to find out security vulnerabilities for the target systems mentioned under “Scope Details” using external Penetration Testing techniques.

Wherever applicable, sample evidences are listed in the “Appendix A – Evidences” section

Scope Details:

S. No	URL
1	http://83.212.174.87/

2. Methodology



2.1 Reconnaissance

The application details were gathered, and the initial planning and analysis performed to perform the penetration testing activity. This includes understanding the target environment, hosting infrastructure etc. through information gathering techniques.

2.2 Port Scanning

This phase uses Port Scanning tools such as Nmap to determine the list of open ports available to the public internet and to build a threat profile based on the services present in the target systems.

2.3 Penetration Testing

Based on the result from both the port scan and reconnaissance an attack profile was planned and executed using Penetration Testing techniques which includes both manual and automated ways to discover vulnerabilities and exploitation possibilities in the target infrastructure. Results from the Penetration Test validated for the presence of False Positives and False Negatives.

2.4 Reporting

Once the activity is completed, a severity is assigned with each finding and documented.

3. Summary of Results

S. No	Vulnerabilities	Severity Rating	Status
1	SQL Injection – Authentication Bypass	Critical	Open
2	Cross Site Scripting	Critical	Open
3	Application uses clear text HTTP protocol	High	Open
4	Clickjacking	Medium	Open

4. Report

4.1 1. SQL Injection – Authentication Bypass

Description:

A SQL injection attack consists of insertion or "injection" of a SQL query via the input data from the client to the application. A successful SQL injection exploit can read sensitive data from the database, modify database data (Insert/Update/Delete), execute administration operations on the database (such as shutdown the DBMS), recover the content of a given file present on the DBMS file system and in some cases issue

commands to the operating system. SQL injection attacks are a type of injection attack, in which SQL commands injected into data-plane input in order to effect the execution of predefined SQL commands. The login page is vulnerable to SQL injection attacks using that an attacker can login to the application without knowing the credentials.

Severity: Critical

URL Affected: http://83.212.174.87/

Evidence: Image 1

Solution:

To prevent SQL Injection attacks is input validation and parametrized queries including prepared statements. The application code should never use the input directly. The developer must sanitize all input, not only web form inputs such as login forms. They must remove potential malicious code elements such as single quotes. It is also a good idea to turn off the visibility of database errors on your production sites. Database errors used with SQL Injection to gain information about your database.

References:

https://cheatsheetseries.owasp.org/cheatsheets/SQL_Injection_Prevention_Cheat_Sheet.html
https://cheatsheetseries.owasp.org/cheatsheets/Query_Parameterization_Cheat_Sheet.html

4.2 2. Cross Site Scripting

Description:

Cross-Site Scripting (XSS) attacks are a type of injection, in which malicious scripts injected into otherwise benign and trusted websites. XSS attacks occur when an attacker uses a web application to send malicious code, generally in the form of a browser side script, to a different end user. Flaws that allow these attacks to succeed are quite widespread and occur anywhere a web application uses input from a user within the output it generates without validating or encoding it.

An attacker can use XSS to send a malicious script to an unsuspecting user. The end user's browser has no way to know that the script should not be trusted, and will execute the script. Because it thinks the script came from a trusted source, the malicious script can access any cookies, session tokens, or other sensitive information retained by the browser and used with that site. These scripts can even rewrite the content of the HTML page.

Severity: Critical

URL Affected: http://83.212.174.87/

Evidence: Image 2

Solution:

In general, effectively preventing XSS vulnerabilities is likely to involve a combination of the following measures:

- Filter input on arrival. At the point where user input received, filter as strictly as possible based on what is expected or valid input.
- Encode data on output. At the point where user-controllable data output in HTTP responses, encode the output to prevent it from interpreted as active content. Depending on the output context, this might require applying combinations of HTML, URL, JavaScript, and CSS encoding.
- Use appropriate response headers. To prevent XSS in HTTP responses that are not intended to contain any HTML or JavaScript, you can use the Content-Type and X-Content-Type-Options headers to ensure that browsers interpret the responses in the way you intend.
- Content Security Policy. As a last line of defense, you can use Content Security Policy (CSP) to reduce the severity of any XSS vulnerabilities that still occur.

References:

https://cheatsheetseries.owasp.org/cheatsheets/Cross_Site_Scripting_Prevention_Cheat_Sheet.html

4.3 Cleartext submission of password

Description:

Some applications transmit passwords over unencrypted connections, making them vulnerable to interception. To exploit this vulnerability, an attacker must be suitably positioned to eavesdrop on the victim's network traffic. This scenario typically occurs when a client communicates with the server over an insecure connection such as public Wi-Fi, or a corporate or home network that is shared with a compromised computer. Common defenses such as switched networks are not sufficient to prevent this. An attacker situated in the user's ISP or the application's hosting infrastructure could also perform this attack. Note that an advanced adversary could potentially target any connection made over the Internet's core infrastructure.

Severity: High

URL Affected: http://83.212.174.87/

Evidence: Not Applicable

Solution:

Applications should use transport-level encryption (TLS) to protect all sensitive communications passing between the client and the server. Communications that should protected include the login mechanism and related functionality, and any functions where sensitive data can be accessed or privileged actions can be

performed. These areas should employ their own session handling mechanism, and the session tokens used should never be transmitted over unencrypted communications. If HTTP cookies are used for transmitting session tokens, then the secure flag should be set to prevent transmission over clear-text HTTP.

References:

Not Applicable.

4.4 Web Application Vulnerable to Clickjacking

Description:

The remote web server does not set an X-Frame-Options response header or a Content-Security-Policy 'frame-ancestors' response header in all content responses. This could potentially expose the site to a clickjacking or UI redress attack, in which an attacker can trick a user into clicking an area of the vulnerable page that is different than what the user perceives the page to be. This can result in a user performing fraudulent or malicious transactions.

Content-Security-Policy (CSP) has been proposed by the W3C Web Application Security Working Group, with increasing support among all major browser vendors, to mitigate clickjacking and other attacks. The 'frame-ancestors' policy directive restricts which sources can embed the protected resource.

Severity: Medium

URL Affected: http://83.212.174.87/

Evidence: Image 3

Solution:

Return the X-Frame-Options or Content-Security-Policy (with the 'frame-ancestors' directive) HTTP header with the page's response. This prevents the page's content from being rendered by another site when using the frame or iframe HTML tags.

References:

https://www.owasp.org/index.php/Clickjacking_Defense_Cheat_Sheet

5. Conclusion:

All the findings should be reviewed and a remediation plan to fix all the vulnerabilities should be drafted. All the associated hardening documents to be reviewed and updated according to the findings to ensure future infrastructure which reflects similar setup should not be vulnerable.

If any of the vulnerabilities cannot be remediated due to business or software dependencies, it should be documented in the risk register with proper owner assigned for the risk. Compensating control should be drafted to ensure the vulnerable software is not exploitable or damageable.

6. Appendix A – Evidences

Image 1: SQL Injection – Authentication Bypass

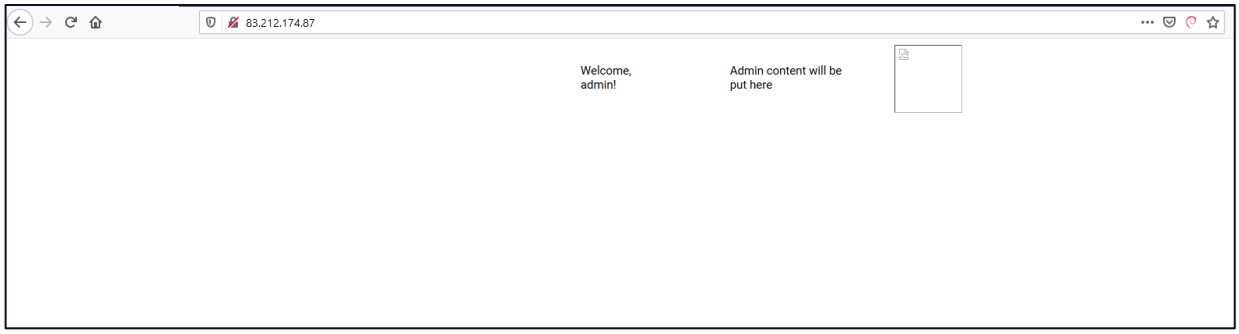


Image 2: Cross-Site Scripting

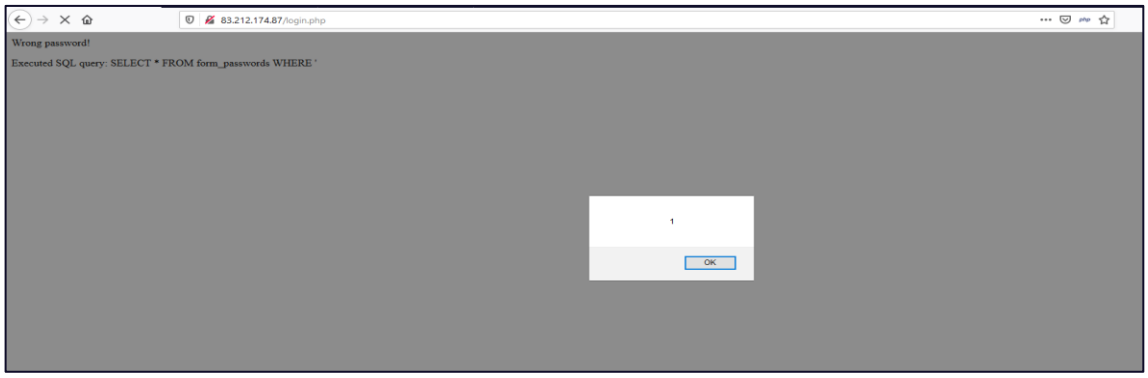


Image 3: Clickjacking