INTRODUCTION

ENGAGEMENT OVERVIEW

Michelle Pantelouris was engaged by Cuppiecake Bakery to conduct a penetration test of their web application hosted within their staging environment.

This type of engagement aims to identify security risks, highlight potential risks, and provide remediation guidance to support an improved security posture.

ENGAGEMENT SCOPE

Based on the agreed objectives, the scope included the **http://192.168.0.120:8080** domain.

Testing was conducted using the user accounts provided by Cuppiecake Bakery.

The usernames for these accounts were 'cake' and 'cake'.

CAVEATS

There were no applicable caveats for this engagement.

EXECUTIVE SUMMARY

SECURITY POSTURE

This section provides an overview of the penetration test completed on the application between the 4th and 10th of March 2024.

The following URL was in scope for the engagement:

- http://192.168.0.120:8080

The following testing approaches were taken for this engagement:

- Simulating an external attack by malicious unauthenticated actors from the Internet.

FINDINGS SUMMARY

Overall, the security posture observed during the test window requires improvement immediately. During testing it was not possible to:

- Gain access to internal network(s).
- Manipulate business logic issues to carry out undesired behaviours within the application.

However, a number of issues were discovered during the testing window. Overall, the consultant discovered one critical risk and one low risk vulnerabilities.

The critical risk that was detected was SQL Injection. This allowed me to obtain the admin credentials, which allowed me to login to the site as an administrator. This is serious as it can lead to confidential information the admin has access to being leaked by hackers, as well as denial of service. To safeguard against SQL injection, utilise parameterised queries, validate input, restrict database permissions, employ ORM libraries, consider database firewalls, conduct regular security audits, encode input/output, and implement secure error handling.

The low-risk vulnerability relates to vulnerable JavaScript dependencies present within the library. It is vulnerable to two CVE. To protect against vulnerable JavaScript dependencies, regularly update dependencies, audit for known vulnerabilities, and restrict the use of dependencies from reputable sources.

RECOMMENDATIONS ROADMAP

MITIGATION PLAN

We've pulled together our recommendations for remediation based on the findings of this engagement.

- Use parameterized queries or prepared statements provided by database APIs to separate SQL code from user input, preventing attackers from injecting malicious SQL commands.
- Audit dependencies using tools like npm audit or yarn audit.

TESTING STRATEGY

It's recommended that the client undertakes a retest after mitigation is taken place.

ISSUE MATRIX

This section lists all issues identified during the assessment, their severity rating, and a brief description along with the associated IDs:

ID	RISK	TITLE	DETAILS
#01	**Critical**	SQL Injection	Attackers can exploit the SQL Injection and gain access to the admin credentials.
#02	**Low**	Vulnerable JavaScript Dependencies	Libraries are vulnerable to CVE's.

TECHNICAL FINDINGS

Throughout this document, we've highlighted a number of findings and have categorised them as Critical, High, Medium, Low or Info Only, explanations for which can be found below.

Our consultants classify risk initially based on three metrics: the CVSS score, the probability of the risk occurring, and the impact should it happen. Using their own real-life experience, common sense is also applied to ensure that we have sensible ratings to help you get a clear understanding of your security posture.

CVSS SCORE

Aka common vulnerability scoring system, a published standard for scoring the severity of a vulnerability.

PROBABILITY

How likely is it that a threat actor can exploit the vulnerability? Do public exploits exist?

IMPACT

The impact rating is how much of an effect a successful exploit could have on your organisation.

RISK RATINGS

CRITICAL	Critical vulnerabilities can lead to direct compromise of the host, allow unauthorised access to sensitive data, or impact the organisation seriously from a financial or reputational perspective
HIGH	A risk rating of 'high' indicates that the vulnerability presents a severe threat to the operation or integrity of the relevant host. Exploitation can lead to the compromise of the host or data, but usually as part of a blended attack
MEDIUM	Vulnerabilities that are medium risk may allow an attacker limited access to the host or data, but they usually require specific conditions for an exploit to be successful, so the risk is considered to be moderate
LOW	Low-risk vulnerabilities may provide an attacker with supplementary information about the organisation (e.g., staff contact information or info about the supporting architecture), so exploitation would be unlikely to have a tangible effect on overall security
INFO ONLY	Findings categorised as 'info only' are those which pose no direct security risk, but the consultant feels that they are noteworthy

SQL Injection Leads to Application Takeover

CRITICAL

CVSS:	IMPACT:	PROBABILITY:	ISSUE REF:
9.0	HIGH	HIGH	#01

WHAT IS SQL INJECTION?

SQL injection is a critical security vulnerability that exploits weaknesses in web applications' handling of SQL queries. It occurs when malicious actors inject SQL code into input fields or query parameters, manipulating the execution of SQL queries. This manipulation can lead to unauthorized access to databases, extraction of sensitive information, modification of data, or even complete compromise of the application's security. SQL injection attacks are prevalent and pose significant risks to organizations, potentially resulting in data breaches, financial losses, and damage to reputation.

Protecting against SQL injection is crucial for several reasons. Firstly, it safeguards sensitive data stored in databases, including personal information, financial records, and proprietary business data. By preventing unauthorized access and manipulation of this data, organizations can maintain compliance with privacy regulations and protect the trust of their customers and stakeholders. Additionally, mitigating SQL injection vulnerabilities enhances the overall security posture of web applications, reducing the risk of cyberattacks and data breaches. It also helps to maintain the integrity and availability of databases, ensuring that critical systems and services remain operational and resilient to malicious exploitation. Ultimately, protecting against SQL injection is essential for preserving the confidentiality, integrity, and availability of data and systems in today's interconnected digital landscape.

SQL Injection was found at /post. I was able to exploit this vulnerability to gain access to the admin credentials and I was able to login.

This is a critical vulnerability because having access to the admin credentials means the hacker has access to everything the admin has access to including all of the confidential information or files. This can then lead to denial of service.

TECHNICAL EVIDENCE

Example Request:

I ran the domain through Burp Suite which found potential injection point.

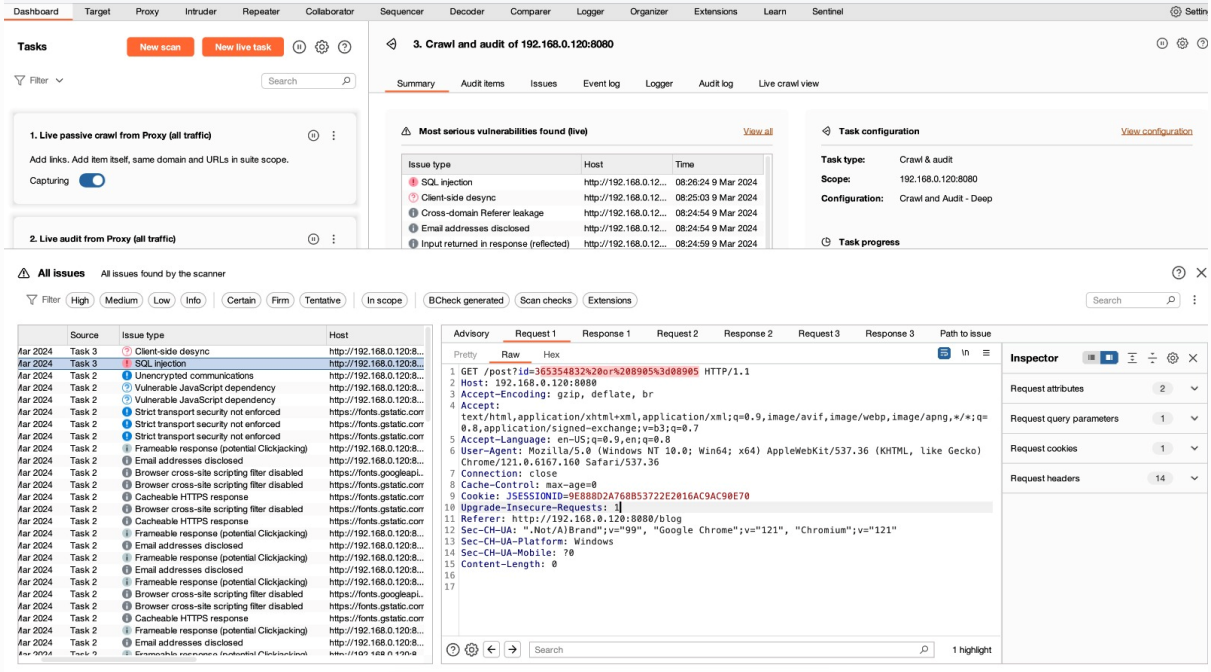


FIGURE 1

I then used SQL map to confirm the SQL Injection was there.

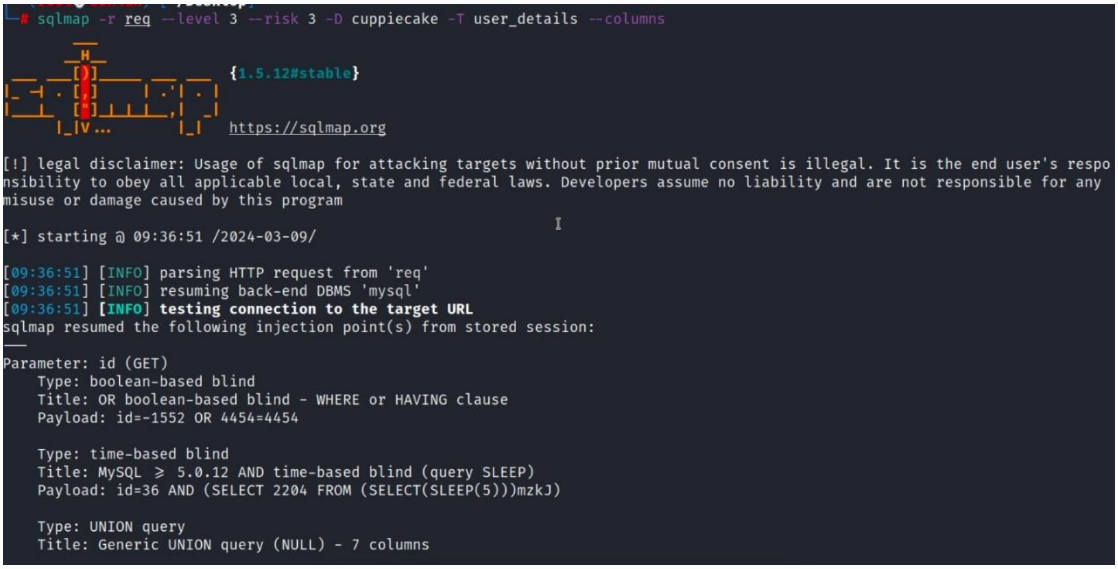


FIGURE 2

I then looked for the columns within the cuppiecup database that I had found. I found two columns that an attacker would look for. Username and Passwords.

```
[09:36:51] [INFO] the back-end DBMS is MySQL
web application technology: JSP
back-end DBMS: MySQL ≥ 5.0.12
[09:36:51] [INFO] fetching columns for table 'user_details' in database 'cuppiecake'
Database: cuppiecake
Table: user_details
[10 columns]
+----------------+---------------+
| Column         | Type          |
+----------------+---------------+
| Active         | varchar(1)    |
| created_by     | varchar(45)   |
| created_date   | timestamp     |
| name           | varchar(100)  |
| userID         | int           |
| usr_email_id   | varchar(45)   |
| usr_mobile_no  | varchar(45)   |
| usr_name       | varchar(45)   |
| usr_password   | varchar(100)  |
| usr_role       | int           |
+----------------+---------------+

[09:36:51] [INFO] fetched data logged to text files under '/root/.local/share/sqlmap/output/192.168.0.120'
[09:36:51] [WARNING] your sqlmap version is outdated

[*] ending @ 09:36:51 /2024-03-09/
```

FIGURE 3

I then dug deeper into the database to find the usernames and passwords within the database. It came back with the admin credentials.

```
[09:38:09] [INFO] the back-end DBMS is MySQL
web application technology: JSP
back-end DBMS: MySQL ≥ 5.0.12
[09:38:09] [INFO] fetching entries of column(s) 'usr_name,usr_password' for table 'user_details' in database 'cuppiecake'
you provided a HTTP Cookie header value, while target URL provides its own cookies within HTTP Set-Cookie header which intersec
t with yours. Do you want to merge them in further requests? [Y/n] y
Database: cuppiecake
Table: user_details
[1 entry]
+----------+--------------+
| usr_name | usr_password |
+----------+--------------+
| admin    | admin@123    |
+----------+--------------+

[09:38:11] [INFO] table 'cuppiecake.user_details' dumped to CSV file '/root/.local/share/sqlmap/output/192.168.0.120/dump/cuppi
ecake/user_details.csv'
[09:38:11] [INFO] fetched data logged to text files under '/root/.local/share/sqlmap/output/192.168.0.120'
[09:38:11] [WARNING] your sqlmap version is outdated

[*] ending @ 09:38:11 /2024-03-09/
```

Figure 4

Next, I went back to the cuppiecup website and logged in with the credentials I found, and discovered that I have admin access.

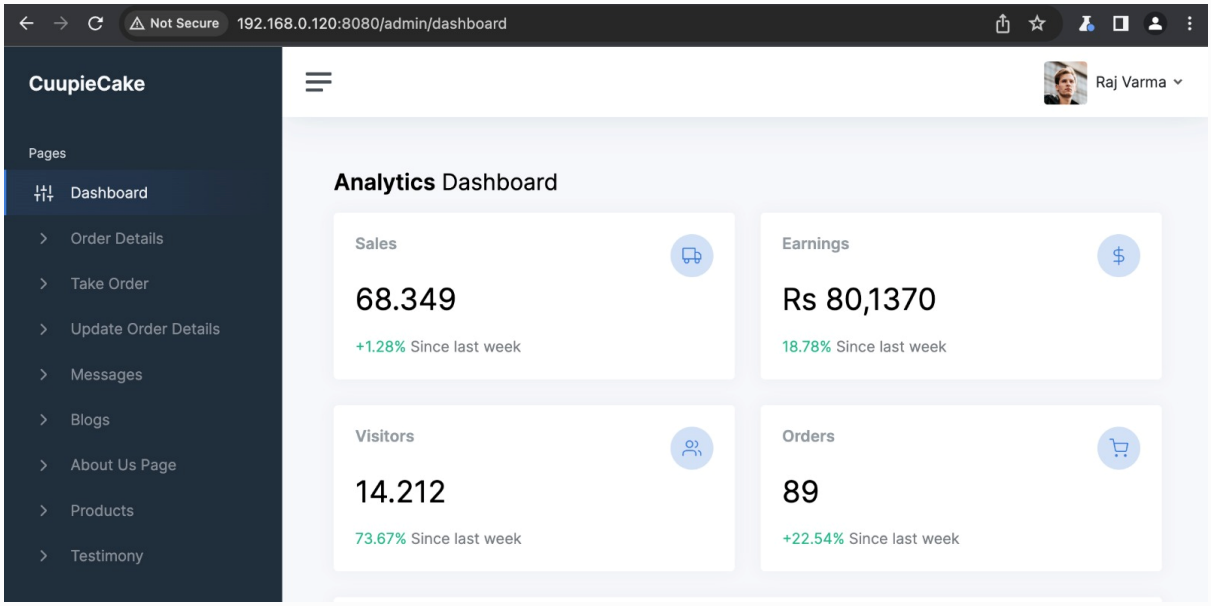


Figure 5

AFFECTED ASSETS

- http://192.168.0.120:8080

REMEDIATION GUIDANCE

Protecting against SQL injection requires implementing various security measures at different layers of the application stack. Here are some best practices to mitigate the risk of SQL injection attacks:

- Use Parameterized Queries or Prepared Statements: Instead of dynamically constructing SQL queries by concatenating user input, use parameterized queries or prepared statements provided by database APIs. These methods separate SQL code from user input, preventing attackers from injecting malicious SQL commands.

- Input Validation and Sanitization: Validate and sanitize all user input before using it in SQL queries. Ensure that input data conforms to expected formats, lengths, and character sets. Use whitelisting approaches to allow only known-safe input and reject or sanitize any input that doesn't meet validation criteria.

- Least Privilege Principle: Restrict database permissions to the minimum level required for application functionality. Avoid granting excessive privileges to application accounts, as this reduces the potential impact of a successful SQL injection attack.

- Use Object Relational Mapping (ORM) Libraries: ORM libraries provide an abstraction layer between application code and the database, automatically handling parameterization and escaping of user input. This reduces the likelihood of SQL injection vulnerabilities in application code.

Vulnerable JavaScript Dependency

Low

CVSS:	IMPACT:	PROBABILITY:	ISSUE REF:
5.5	Low	HIGH	#02

WHAT IS VULNERABLE JAVASCRIPT DEPENDENCY?

A vulnerable JavaScript dependency is a component, library, or module used in a JavaScript-based application that contains security flaws or weaknesses. These vulnerabilities could potentially be exploited by attackers to compromise the security of the application, steal sensitive data, or disrupt its normal operation.

Here's why it's crucial to protect against vulnerable JavaScript dependencies:

Security Risks: Vulnerable dependencies can introduce security risks into the application. Exploiting these vulnerabilities, attackers can launch various types of attacks, such as Cross-Site Scripting (XSS), Cross-Site Request Forgery (CSRF), or Remote Code Execution (RCE).

Data Breaches: If attackers exploit vulnerabilities in JavaScript dependencies, they may gain unauthorized access to sensitive data stored within the application or accessed by it. This could lead to data breaches, exposing users' personal information, financial data, or other confidential data.

Compromise of User Trust: Security breaches resulting from vulnerable dependencies can erode user trust in the application or the organization behind it. Users expect their data to be handled securely, and breaches can damage the reputation of the application or brand, leading to loss of customers and revenue.

Legal and Regulatory Compliance: Many industries are subject to strict regulations regarding data protection and privacy. Failure to protect against vulnerabilities in JavaScript dependencies can lead to non-compliance with these regulations, resulting in legal consequences, fines, or other sanctions.

Operational Disruption: Exploitation of vulnerabilities in JavaScript dependencies can disrupt the normal operation of the application, causing downtime, service outages, or performance degradation. This can impact productivity, revenue generation, and customer satisfaction.

To mitigate the risks associated with vulnerable JavaScript dependencies, it's essential to:

Regularly update dependencies to their latest secure versions.
Monitor security advisories and vulnerability databases for known issues.
Use security scanning tools to identify and remediate vulnerabilities in dependencies.

Restrict the use of third-party libraries to trusted sources and review code before integration.
Implement secure coding practices and follow security guidelines provided by library maintainers.

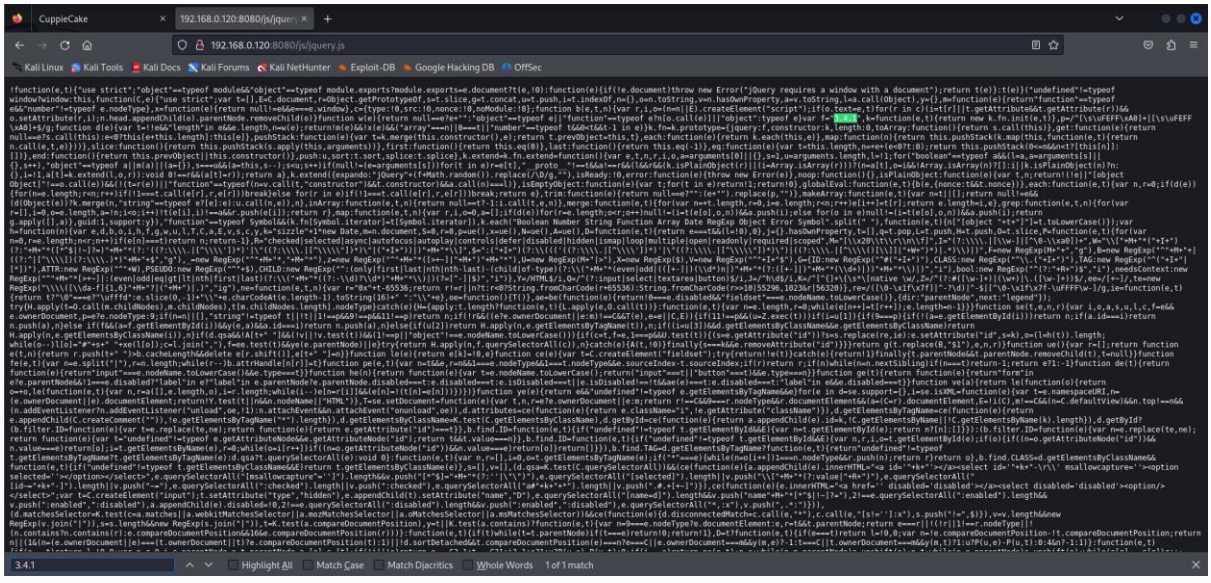
By prioritizing the protection against vulnerable JavaScript dependencies, organizations can strengthen the security posture of their applications, safeguard sensitive data, and maintain user trust and compliance with regulatory requirements.

TECHNICAL EVIDENCE

The vulnerable library was found in /js/jquery.js. The jquery version is 3.4.1. Two CVE's were found within the library.

The first was CVE-2020-11023. In jQuery versions greater than or equal to 1.0.3 and before 3.5.0, passing HTML containing <option> elements from untrusted sources - even after sanitizing it - to one of jQuery's DOM manipulation methods (i.e. .html(), .append(), and others) may execute untrusted code. This problem is patched in jQuery 3.5.0.

The second as CVE-2020-11022. In jQuery versions greater than or equal to 1.2 and before 3.5.0, passing HTML from untrusted sources - even after sanitizing it - to one of jQuery's DOM manipulation methods (i.e. .html(), .append(), and others) may execute untrusted code. This problem is patched in jQuery 3.5.0.



AFFECTED ASSETS

- http://192.168.0.120:8080/js/jquery.js

REMEDIATION GUIDANCE

To protect against vulnerable JavaScript dependencies:

- Keep dependencies updated regularly to their latest secure versions.
- Audit dependencies using tools like npm audit or yarn audit.
- Stay informed about security advisories related to dependencies.
- Minimize the number of dependencies and choose reputable sources.
- Pin dependency versions to prevent unintended upgrades to vulnerable versions.
- Conduct peer code reviews to catch vulnerabilities early.
- Use static code analysis tools to detect security issues.
- Maintain a list of approved dependencies and versions.
- Implement security headers like Content Security Policy.
- Provide security training for developers to promote secure coding practices.

METHODOLOGY

WEB APPLICATION TESTING METHODOLOGY

I based my application testing methodology on the OWASP framework. It's a recognised industry standard used across the world by all the best security testing businesses.

Testing included a full audit of your application and included coverage of the following key security areas:

- Web server and supporting infrastructure configuration review.
- Application mapping
- Encryption / cryptography review
- Session handling review
- Input validation review
- Application logic review
- Information leakage review
- Overall application code quality
- Access control checks
- Environment / configuration / integration web server issues
- Authentication review
- API security review (where applicable)