

COMP3911 Coursework

Group members:

Andreas Athanasiadis - 201536750

Siddesh R Ohri - 201659113

Laleendteshwaran Vigneshwaran - 201617325

Varshith Ankireddypalle – 201578812

Ziyi Li – 201587206

Contents

Analysis of Flaws	2
SQL Injection	2
Lack of HTTPS	2
Plain-Text Password Storage	3
Resource Exhaustion (DOS)	3
Entity-Based Access Control (EBAC)	3
Attack Tree	5
Fix Identification	6
Fixes Implemented	7
References	9
Appendix	9

Analysis of Flaws

List of five flaws we have identified:

1. **SQL Injection:** Unsanitized inputs are directly interpolated into SQL queries, allowing attackers to manipulate the *database*.
2. **Lack of HTTPS:** The application does not use HTTPS, exposing user data to interception and manipulation during transmission.
3. **Plain-Text Password Storage:** Passwords are stored in plain text instead of being securely hashed, exposing them to compromise if the database is breached.
4. **Resource Exhaustion (DOS):** Unrestricted database queries can be exploited to overload the system, causing denial of service.
5. **Entity-Based Access Control:** There is no access control to restrict patient records to only their assigned GP, leading to privacy violations.

SQL Injection

The application uses raw SQL query strings and lacks input validation, making it vulnerable to SQL injection attacks. Unsanitized user inputs are directly integrated into SQL queries, allowing attackers to manipulate the database. This flaw was identified through a manual code review and confirmed through dynamic testing, where inputs like `1=1--` bypassed authentication and granted unauthorized access. Also, no input sanitization and no form validation was noticed allowing users to submit anything they wanted.

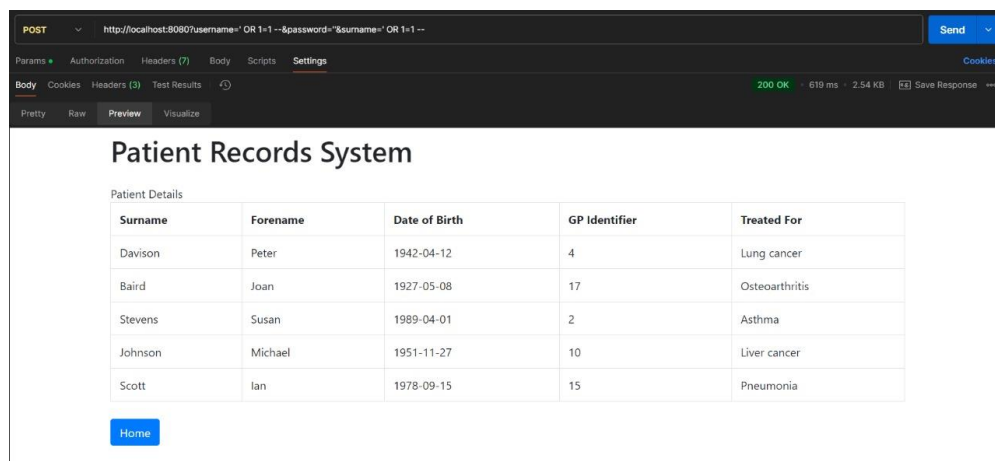


Figure 1: All records accessed through Injection

Lack of HTTPS

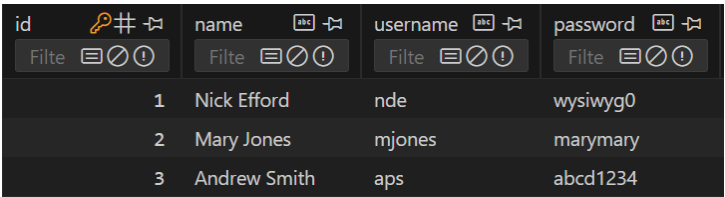
The lack of HTTPS in the application exposes it to significant security vulnerabilities, enables attackers to intercept, modify, or steal sensitive data. During the analysis, it was discovered that the application was using HTTP instead of HTTPS, which was evident from the URL not having "https://" and confirmed by inspecting the network traffic. This flaw makes the application

susceptible to man-in-the-middle attacks, where attackers could intercept and steal sensitive information. Implementing HTTPS would encrypt the communication, safeguarding user data from such attacks.

Plain-Text Password Storage

The application stores user passwords in plaintext in the database, making them highly vulnerable to exposure in the event of a data breach. Secure applications should employ hashing techniques to protect user credentials. During a database review, it was discovered that passwords were stored in plaintext without any encryption or hashing.

Further investigation of the authentication process confirmed that passwords were directly compared without any hashing, indicating a lack of secure password management practices. Additionally, the absence of logging mechanisms to track database access or failed login attempts was noted, which could have helped detect potential security incidents.

A screenshot of a database table with four columns: id, name, username, and password. Each column has a filter icon above it. The table contains three rows of data. The first row shows id 1, name Nick Efford, username nde, and password wysiwyg0. The second row shows id 2, name Mary Jones, username mjones, and password marymary. The third row shows id 3, name Andrew Smith, username aps, and password abcd1234.

id	name	username	password
1	Nick Efford	nde	wysiwyg0
2	Mary Jones	mjones	marymary
3	Andrew Smith	aps	abcd1234

Figure 2: Plain text credentials

Resource Exhaustion (DOS)

The application’s design flaw permits limitless database queries, including extensive searches without constraints. This vulnerability could allow users or attackers to exploit the system, potentially exhausting resources, causing slowdowns, or leading to crashes, thereby creating a denial-of-service (DoS) condition [Learning Center, n.d.].

While we did not directly test this issue, as the application was deployed locally and lacked a production-like environment for realistic stress testing, our analysis was inspired by an article discussing similar vulnerabilities. The article highlighted how unrestricted database queries could result in significant resource strain, motivating us to evaluate the potential impact of this flaw theoretically.

Entity-Based Access Control (EBAC)

The program does not have a entity-based access control (EBAC) system to limit access to confidential patient records. This lapse permits any authenticated General Practitioner (GP) to access all patient records, contravening privacy norms and regulatory mandates.

Access assessments were performed utilising GP accounts with diverse roles. It was noted that general practitioners may get and access patient records, including those not pertaining to their designated patients.

The identified defects undermine the system's security and integrity, requiring prompt remediation to prevent exploitation and assure adherence to data protection rules.

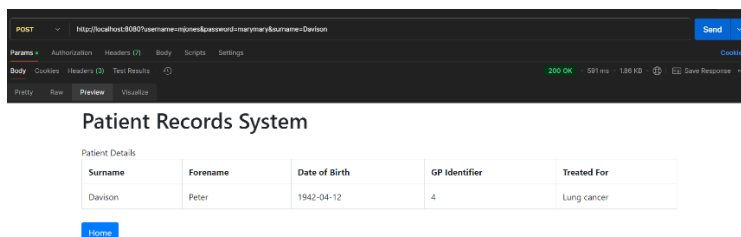


Figure 3: Patient record accessed by mjohnes

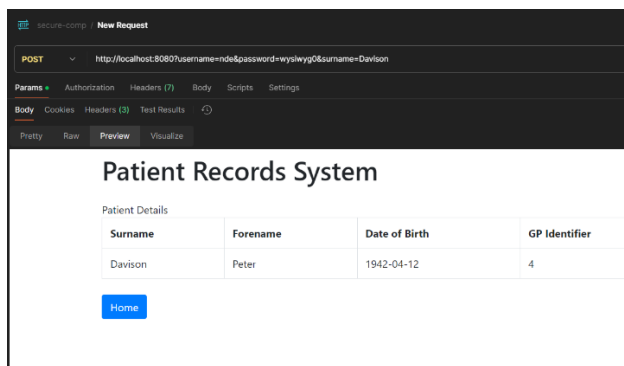


Figure 4: Patient record accessed by nde

Flaw	Spoofing	Tampering	Repudiation	Information Disclosure	Denial of Service (DoS)	Elevation of Privilege
SQL Injection	✓	✓	✓	✓	✓	✓
Lack of HTTPS	✓	✓		✓	✓	✓
Plain-Text Password Storage	✓	✓	✓	✓		✓
Resource Exhaustion (DOS)					✓	
Patient Records Accessed by Every GP		✓	✓	✓		✓

STRIDE Table per flaw

Attack Tree

The attack tree is organized hierarchically, starting with "**Unauthorized Access to Patient Data**" as the root node, representing the ultimate goal of an attacker exploiting the vulnerable system. It branches into several major attack vectors through which an attacker can exploit the system:

- **SQL Injection** includes bypassing authentication and accessing or manipulating the database.
- **Lack of HTTPS** enables attackers to intercept, modify, or steal sensitive data
- **Credential Acquisition** covers obtaining unhashed credentials from the database or tricking users into disclosing passwords.
- **Resource Exhaustion (DoS)** focuses on targeting rate-limited endpoints to disrupt system functionality.
- **EBAC Flaws** highlight scenarios like attackers compromising GP credentials to gain access to multiple patient records.

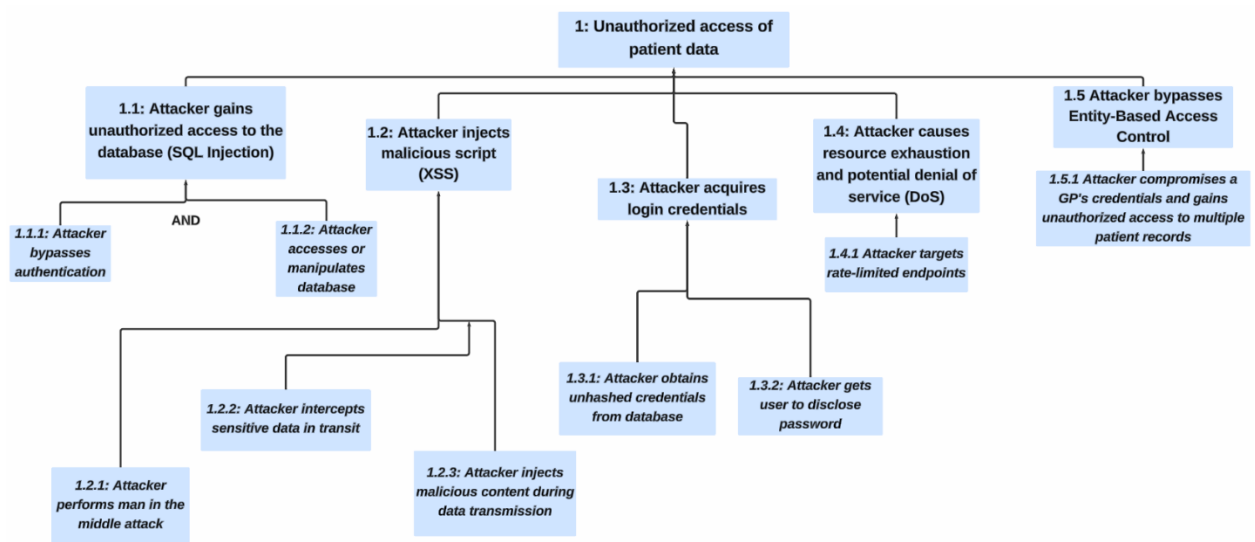


Figure 5: Attack Tree

Assumptions:

Patients have only one GP that monitors them.

Fix Identification

The fixes implemented address the vulnerabilities outlined in the attack tree, systematically reducing the risk of unauthorized access to patient data by targeting critical nodes:

1. SQL Injection

Reason: This fix directly addresses the root vulnerability in Node 1.1, "Attacker gains unauthorized access to the database." By implementing parameterized queries, user inputs are no longer directly interpolated into SQL commands. This prevents attackers from bypassing authentication (Node 1.1.1) and gaining control over or manipulating the database (Node 1.1.2). As a result, unauthorized database access through SQL Injection is eliminated.

2. Lack of HTTPS

Reason: The fix mitigates the vulnerability in Node 1.2, "Lack of HTTPS," by implementing secure HTTPS encryption for all communications. This ensures that sensitive data, such as login credentials, is transmitted securely, protecting it from interception or modification by attackers through man-in-the-middle (MITM) attacks. It strengthens the application's defences by safeguarding user data during transmission and preventing unauthorized access or tampering.

3. Plain-text Password Storage

Reason: Hashing passwords tackles the vulnerability in Node 1.3.1, "Attacker obtains unhashed credentials from the database." By replacing plaintext password storage with secure hashing, even if the database is compromised, attackers cannot directly exploit the credentials. This limits the risk of unauthorized access through credential-based attacks (Node 1.3). The hashed passwords significantly enhance the resilience of user authentication.

4. Resource Exhaustion (DoS)

Reason: Rate-limiting counteracts the vulnerability in Node 1.4, "Attacker causes resource exhaustion and potential denial of service." By limiting database queries and managing the data retrieval process more efficiently, these measures prevent attackers from overwhelming the system (Node 1.4.1). This ensures consistent application performance and availability, even under potentially malicious usage patterns.

5. Entity-Based Access Control (EBAC) Flaws

Reason: Addressing Node 1.5, "Attacker compromises GP credentials to access multiple patient records," this fix enforces stricter access control policies. By associating GPs with specific patient records and validating roles during queries, the application ensures that users can only access data they are authorized to view. This eliminates the risk of broad access through compromised GP credentials and ensures compliance with data privacy regulations.

Each fix strengthens the application's security by addressing the specific attack vectors outlined in the attack tree, providing a robust defence against threats targeting patient data.

Fixes Implemented

Each fix ensures the system is more robust against the associated security threats.

1. SQL Injection

Fix: Replaced dynamic SQL queries with parameterized queries using prepared statements, implemented input sanitization, form validation, and configured a strict Content Security Policy (CSP) in meta tags.

Implementation Details: All user inputs interacting with the database are now processed using parameterized queries, which separate SQL logic from user data, preventing malicious input from altering query execution. Input sanitization and form validation are applied to ensure only valid data is accepted, rejecting any special characters or malformed input that could be used for SQL injection. The Content Security Policy (CSP) in meta tags is configured to restrict the types of content that can be executed on the page, preventing external scripts from being loaded that could manipulate user inputs or data.

Impact: This fix prevents attackers from injecting malicious SQL code into database queries. For instance, attempts to bypass login authentication by inserting " OR 1=1; --" into the username field are now blocked due to the combination of input validation, parameterized queries, and sanitization. This significantly reduces the risk of unauthorized access or data manipulation via SQL injection attacks.

2. Lack of HTTPS

Fix: Implemented HTTPS with SSL/TLS encryption by configuring a valid certificate, enforcing HTTPS redirection, and replacing HTTP links with HTTPS in the application (View [Appendix](#)).

Implementation Details: A valid SSL/TLS certificate was configured to enable secure communication between the client and server. Additionally, HTTP to HTTPS redirection was enforced to ensure all requests are securely handled.

Impact: These changes ensure that all data transmitted between the client and server is encrypted, protecting it from interception or modification during transit mitigating risks from man-in-the-middle (MITM) attacks.

3. Plain-text Password Storage

Fix: Transitioned from storing plain-text passwords to using secure password hashing with a strong algorithm (e.g., SHA-256).

Implementation Details: Passwords are hashed before being stored in the database, and all password comparisons are performed securely against the hash.

Impact: Passwords are now securely hashed, rendering them unreadable even if the database is compromised. A database query now shows hashed passwords instead of plaintext, ensuring user credentials are protected.

4. Resource Exhaustion (DoS)

Fix: Added rate-limiting mechanisms for logging in. Configured timeouts and number of attempts per user for database queries to prevent resource overload.

Implementation Details: Implemented a threshold for the number of requests that can be made by a single user within a specific time.

Impact: These measures prevent attackers from overloading the database with excessive queries. Testing demonstrated that attempts to flood the server with wildcard queries no longer cause significant performance degradation.

5. Entity-Based Access Control (EBAC) Flaws

Fix: Introduced EBAC to restrict access to patient records, ensuring GPs can only access the data relevant to their assigned patients.

Implementation Details: Each GP is assigned an ID. Checks are implemented at the API and database levels to enforce restrictions.

Impact: GPs can now only access records of their assigned patients. Attempts to query unauthorized patient records result in access denial. This fix ensures compliance with privacy regulations and significantly reduces the risk of privacy breaches.

6. Logging as a Common Fix

Logging mechanisms have been integrated into the application to enhance visibility and detect suspicious activities across all identified vulnerabilities. These logs are designed to:

- **Monitor for Unauthorized Activities:** Capture failed login attempts, unusual query patterns, and invalid input data.
- **Provide Audit Trails:** Track sensitive actions, such as access to patient data, to support forensic investigations.
- **Improve Detection:** Enable administrators to proactively identify patterns indicative of security threats, such as brute force attacks or resource exhaustion attempts.

Each of these fixes was thoroughly tested to ensure that the application functions as intended while eliminating the identified vulnerabilities. Together, these changes enhance the security and reliability of the application.

References

Learning Center. (n.d.). *What is Rate Limiting | Types & Algorithms | Imperva*. [online] Available at: <https://www.imperva.com/learn/application-security/rate-limiting/>. [Accessed on 25/11/24]

Appendix

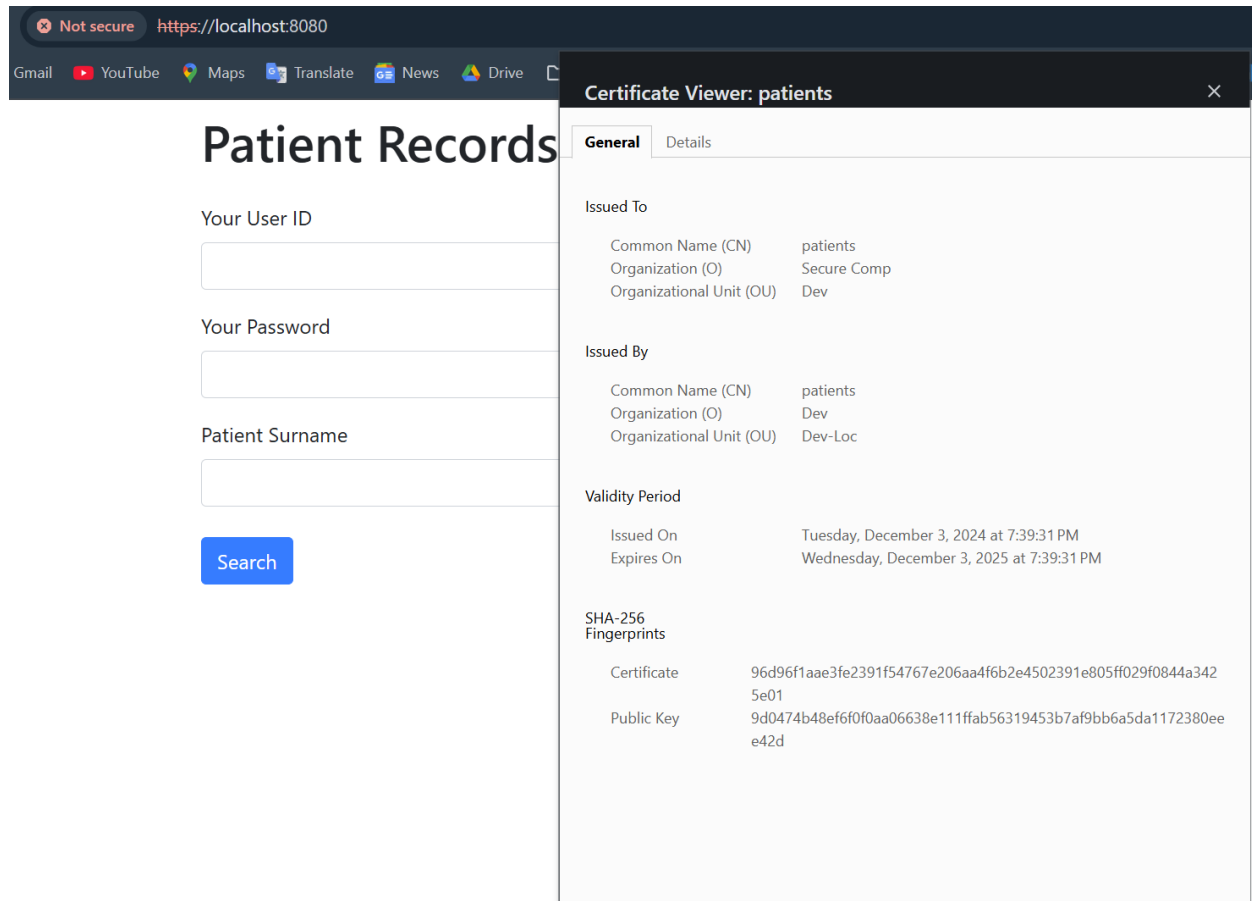


Figure 6: Implementation of HTTPS