SUPERVISOR: AHMED HANIF

NARWAL AUTH API DELIVERABLE 3

MUHAMMAD NASEEM
SAMI NAEEM
HASSAAN FAROOQ

RISKS

Format: (risk numbering. First bullet-point = risk, Second bullet-point = mitigation solution)

1. **Implementation Errors**
    - Errors in implementing cryptographic protocols can create vulnerabilities, compromising the system's security. These mistakes could include improper key management, insufficient randomness, or incorrect hashing techniques.
    - Adhering to established cryptographic standards, like ‘AES’ and ‘RSA’, is essential to mitigate these risks.

2. **Key Management Failures**
    - Proper key management is crucial. If keys are not rotated regularly or replaced in emergencies, the system's security could be compromised.
    - Robust key management procedures are necessary to prevent unauthorized access and data breaches.

3. **Server Overload**
    - High traffic could overwhelm the server, causing it to become unavailable. This might result from legitimate usage spikes or malicious brute force attacks.
    - Implementing load balancing and rate limiting can help manage traffic and maintain server availability.

4. **Cryptographic Library Issues**
    - Using outdated or unmaintained cryptographic libraries can introduce risks. It's important to use well-maintained and tested libraries to ensure system security.
    - Regular updates and vulnerability monitoring are critical.

5. **Network Vulnerabilities**
    - Unsecured communication between client and server can be intercepted by attackers, leading to data breaches.
    - Encrypting data transmissions with protocols like TLS is essential to protect against these vulnerabilities.

6. **User Interface Design Flaws**:
    - A poorly designed user interface can lead to errors and security weaknesses, such as revealing too much information in error messages.
    - An intuitive and secure UI design is crucial.

7. **Insufficient Logging and Monitoring**:
    - Without proper logging and monitoring, detecting and responding to security incidents can be difficult.
    - Comprehensive logging and real-time monitoring are necessary to identify and address threats promptly.

8. **Data Storage Insecurities**:
    - Insecure storage of sensitive data, like hashed passwords, can lead to breaches.
    - Ensuring all sensitive data is encrypted and securely stored.

9. **Inadequate Testing**:
    - Failure to thoroughly test the system can leave it vulnerable to undetected flaws.
    - Rigorous testing, including penetration testing and code reviews.

10. **Third-Party Dependencies**:
    - Dependencies on third-party libraries and services can introduce risks if they are compromised.
    - Regularly reviewing, updating, and monitoring third-party components.

11. **Regulatory Compliance**:
    - Ensuring compliance with data protection regulations like GDPR or CCPA is essential to avoid legal issues.
    - Implementing measures to protect user data and maintaining regulatory compliance.

Technology Stack

After a thorough assessment of our requirements and resources we've decided to work on our project using the following technology stack:

Back-end: Python(Flask)

- Python is easy to prototype with and has excellent cryptography libraries like 'cryptography' and 'PyCryptodome' which are essential for implementing our zero-knowledge proofs securely. Additionally, Flask is lightweight and quick to set up.
- For an expanded user base, we can either multi-thread by load balancing across multiple Flask instances to distribute user requests or rewrite our code in Node.js and Express.js This combination allows us to develop rapidly while still keeping our project scalable if it goes public
- Advantages and disadvantages of Python (Flask):
    - **Advantages:**
    - The main advantage of Python(Flask) is that it is easy to prototype and rapid development. It has a wide range of cryptography libraries available.
    - **Disadvantages:**
    - Scaling requires load balancing across multiple instances which requires an additional setup. Or we might need to rewrite for very large user bases.
- Advantages and disadvantages of Node.js with Express js
    - **Advantages:** It offers better performance for I/O bound operations due to non-blocking, event-driven architecture. As it can handle a large number of concurrent connections efficiently.
    - **Disadvantages:** It has few libraries as compared to Python. Furthermore, it can be more complex to set up initially compared to Flask

Front-end: React or HTML/CSS + Javascript

- React offers better scalability for larger applications. It has a Virtual DOM, which allows for efficient updates by rendering only the changed components, saving a lot of memory. Furthermore, it offers asynchronous operation because Node.js supports single-threaded programming, allowing other operations to continue.
- Meanwhile, using HTML/CSS + JavaScript can be effective for basic implementations and provide direct control over the DOM. Additionally, it is lightweight.

Database: MySQL-connector-python or MYSQL or SQLite

- Mysql-connector-python: it integrates with Flask allowing straightforward database interactions. It's also reliable for large and small-scale applications. It ensures robust and scalable data storage.

- MYSQL: is a widely used, reliable database system.it is suitable for read-heavy and write-heavy operations. It can handle large datasets and high-traffic
- SQLite: SQLite is an attractive option because it is serverless, self-contained, and requires zero configuration, making it incredibly easy to set up and use. SQLite’s simplicity is ideal for initial phase projects where the database size and user load are moderate, and its portability allows for easy migration to more robust SQL databases if needed.

Overview and Advantages:

Our chosen technology stack enables us to develop quickly and iterate rapidly, which is crucial for the initial phase of our project. This stack streamlines the development process with minimal configuration, allowing my team and me to concentrate on implementing the core features. The selected technologies equip us with all the necessary tools to effectively demonstrate the zero-knowledge authentication concept and manage a moderate user load if the demo goes public. Additionally, the stack is adaptable and can be scaled or transitioned to more robust technologies as the project evolves. By leveraging established libraries and frameworks, we ensure that we adhere to security best practices from the very start.

For cryptographic operations on the backend, we utilize Python's cryptography library. This library offers a comprehensive range of cryptographic recipes and primitives, guaranteeing secure implementations. It is actively maintained and frequently updated, ensuring we benefit from the latest security enhancements and bug fixes. On the frontend, we leverage the Web Crypto API for modern browsers to perform efficient and secure cryptographic operations natively. For older browsers, we rely on crypto-js, a well-established JavaScript library for cryptographic functions. Additionally, we use big-integer.js on the frontend to handle large numbers, which is essential for cryptographic calculations in zero-knowledge proofs. This library ensures efficient arithmetic operations and consistent performance across different browsers.