Objective:

Your task is to create a report for senior management, either in Google Docs or Google Slides, listing each of the five vulnerabilities, and tying them to the OWASP Top 10. In addition, you must document the risk posed by each one, the severity, and how to remediate them. Be sure to provide justification for the severity rating.

● Cross-site scripting (XSS)

https://owasp.org/www-project-top-ten/2017/A7_2017-Cross-Site_Scripting_(XSS)
A7:2017-Cross-Site Scripting (XSS)

● Cross-site request forgery (CSRF)

https://blog.nvisium.com/p139#:~:text=Incidence%20Rate%20%20%20%20Title%20%20,%20%2018.36%25%20%2027%20more%20rows%20
A6 - sensitive data exposure
A7 - missing function level access control
A9 - using known vulnerable components

● SQL injection

A1:2017-Injection

● Sensitive data exposure

A3:2017-Sensitive Data Exposure

● Insufficient logging and monitoring

A10:2017-Insufficient Logging & Monitoring

OWASP 10

A1:2017-Injection

risk posed

Injection can lead to data loss, corruption, or disclosure to unauthorized parties, as well as loss of responsibility and access prohibition. Injection can occasionally result in the total control of the host.
The business effect is determined by the application's and data's requirements.

Remediation

- In order to avoid injection, data must be kept distinct from instructions and queries. Using a safe API, which avoids using the interpreter altogether or offers a parameterized interface, or migrating to Object Relational Mapping Tools, is the preferable choice (ORMs).
- Note that even if stored procedures are parameterized, they can still introduce SQL injection if PL/SQL or T-SQL restriction enzyme queries and data, or executes hostile data using EXECUTE IMMEDIATE or exec ().
- Use server-side input validation that is affirmative or "whitelisted." Many applications, such as text fields or APIs for mobile applications, require special characters, thus this isn't a full protection.
- For any remaining dynamic queries, use the interpreter's unique escape syntax to escape special characters. To avoid bulk exposure of records in the event of SQL injection, use LIMIT and other SQL restrictions within queries.

A3:2017-Sensitive Data Exposure

risk posed

Failure frequently exposes all data that was supposed to be safe. Sensitive personal information (PII) data, such as health records, credentials, personal data, and credit cards, is commonly included in this data, and it is usually subject to protection under rules or policies such as the EU GDPR or local privacy laws.

Remediatio

At a least, do the following tasks and review the references:

- Classify the data that an application processes, stores, or transmits. Determine which information is sensitive in light of privacy rules, regulatory obligations, or commercial considerations.
- Implement controls in accordance with the categorization.
- Don't keep critical information around needlessly. It should be discarded as quickly as feasible, or it should be tokenized or truncated in accordance with PCI DSS. It is impossible to steal data that is not saved.
- Make sure all sensitive data is encrypted at rest.

- Use effective key management and ensure that up-to-date and robust standard algorithms, protocols, and keys are in place.
- Use secure protocols like TLS with perfect forward secrecy (PFS) ciphers, server cipher priority, and secure parameters to encrypt all data in transit. Use HTTP Strict Transport Security directives to enforce encryption (HSTS).
- Store passwords using strong adaptable and encrypted hashing algorithms with a work factor (delay factor), such as Argon2, scrypt, bcrypt, or PBKDF2. * Disable caching for sensitive data responses. Check the efficacy of the setup and settings on your own.

A6 - sensitive data exposure

risk posed

provide unauthorized access to certain system data or functionality to attackers A full system compromise is feasible.

Remediation

Implement secure installation methods, such as: * A repeatable hardening process for rapid deployment of a new environment with adequate security. In each environment, separate credentials are utilized (this process should be automated).

- a stripped-down platform with no extra features, components, or documentation Unused features and frameworks should be removed or not installed.
- As part of the patch management process, review and update settings relevant to all security notes, updates, and patches (e.g. A9:2017). Check the permissions for cloud storage (e.g. S3 bucket permissions).
- Use segmentation, containerization, or cloud security groups to protect the separation of components or tenants in an application's architecture (ACLs).
- Send clients security directives (e.g. Security Headers).
- automate the verification of all setups and settings' efficacy.

A7:2017-Cross-Site Scripting (XSS)

risk posed

DOM XSS, impact mild;
stored XSS, impact severe,
including remote code execution on the victim's browser, such as stealing passwords, sessions, or sending malware to the user.

Remediation

Separating untrusted data from active browser content is required to prevent XSS. This may be accomplished by: * Using frameworks that are

designed to automatically avoid XSS (e.g., latest Ruby on Rails, and React JS). Learn the XSS protection limits of each framework and how to manage use cases that aren't addressed.

1. * Using HTML output information to escape unsafe HTTP request data
    1.1. Body
    1.2. Attribute
    1.3. JavaScript
    1.4. CSS
    1.5. URL
    1.6. The essential data escape strategies are detailed in the OWASP Cheat Sheet 'XSS Prevention.'
2. * When changing the browser page on the client side, use context-sensitive encoding to prevent DOM XSS. Similar user centric escape strategies, as outlined in the OWASP Cheat Sheet 'DOM based XSS Mitigation,' can be applied to browser APIs if this cannot be prevented.
3. * Using a Content Security Policy (CSP) as a defense-in-depth XSS mitigation mechanism. It's effective assuming there aren't any additional flaws which would allow malicious code to be installed via local file inclusions (e.g. path traversal overwrites or vulnerable libraries from permitted content delivery networks).

A9:2017-Using Components with Known Vulnerabilities

risk posed

While some known vulnerabilities have relatively modest consequences, known flaws in components have been exploited in some of the greatest breaches to date. This danger may or may not be at the top of your list, depending on the assets you're safeguarding.

Remediation

use a patch management to:

- Remove unneeded dependencies, superfluous features, components, files, and documentation using a patch management system. Inventory the versions of client and server-side components and dependencies on a regular basis. Continuously monitor sources such as CVE and NVD for component vulnerabilities. Make use of technologies for analyzing program composition. Implement security vulnerability email notifications.
- Only use secure connections to acquire components from official sources. Keep an eye out for libraries and components that haven't been updated in a while or don't have security updates for older versions.

A10:2017-Insufficient Logging & Monitoring

risk posed

Vulnerability probing is the starting point for the majority of successful attacks. Allowing such probes to run can increase the chances of a successful exploit to nearly 100%.

Detecting a breach took an average of 191 days in 2016, giving plenty of time for harm to be done.

Remediation

- Ensure that all login, access control, and server-side input validation failures can be logged with sufficient user context to identify suspicious or malicious accounts, and held for sufficient time to allow delayed forensic analysis, according to the risk of the data stored or processed by the application.
- Ensure that logs are generated in a format that can be easily consumed by a centralized forensic analysis system.
- Establish or implement an incident response and recovery strategy, such as NIST 800-61 rev 2 or later, to ensure that suspicious actions are recognized and reacted to in a timely manner.
- There are free open - source application protection frameworks like OWASP AppSensor (old wiki), web application firewalls like ModSecurity with the OWASP ModSecurity Core Rule Set, and log correlation software with custom display screens and alerting, as well as log correlation software with custom control panels and alerting.