**Name: Lakshmi Likitha Malemarpuram**

**Reg no: 21BCE8963**

**Email:** [**likitha.21bce8963@vitapstudent.ac.in**](mailto:likitha.21bce8963@vitapstudent.ac.in)

**DATE: 28-08-2023**

**DAY: MONDAY**

**TASK 3: 10 Web Application Attacks**

1. Broken access control

Access control limits what users can access, restricting them to resources within their assigned permissions. [Access control failure](https://owasp.org/Top10/A01_2021-Broken_Access_Control/) commonly results in users performing business functions that require different permissions than they were assigned, among other activities. Failure also leads to unauthorized information disclosure, modification, or data destruction.

Access control vulnerabilities include:

* Violating the principle of least privilege by giving all users access to resources intended for specific roles, users, or permissions groups.
* Bypassing access control checks through URL modification, internal application state modification, or HTML page modification. An API attack tool can also be used to bypass access control checks.
* Missing API access controls for PUT, POST, or DELETE.
* Elevation of privilege, where an attacker has access as a user without logging in or can perform admin-level functions from a lower-privilege user account.

Prevention

Limit access to resources with access control. Access control works in a trusted server environment where data cannot be modified by the attacker. Protection methods include:

* Access should be denied by default, unless the item is a public resource.
* Reuse access control mechanisms throughout the application
* Enforce application business limits
* Disable directory listings for web servers
* Rate-limit API and controller access

2. Cryptographic failures

[Cryptographic failures](https://owasp.org/Top10/A02_2021-Cryptographic_Failures/) are a broad symptom of a breakdown or deficiency in cryptography, which can lead to system compromise or sensitive data exposure. Personally identifiable data and credit card numbers are among the data types that require extra protection. Data protection methods are determined by the type of data and whether or not it is subject to data privacy laws such as the EU [General Data Protection Regulation (GDPR)](https://vmblog.com/archive/2022/05/25/4-years-of-gdpr-expert-commentary-shared.aspx#.Y1UhVnZBy3A).

Cryptographic failures include:

* Unenforced encryption in the browser where HTTP security headers are missing.
* Invalid server certificate trust chain.
* Data transmitted in clear text over browser protocols such as FTP, SMTP, and HTTP.
* Deploying weak cryptographic algorithms and protocols; ignoring weak cryptographic algorithms in legacy code.

Prevention

Preventing cryptographic failure depends on application functionality and the type of data used. There are a wide range of aspects to protecting data appropriately. Prevention includes the following (and much more which is covered in OWASP reference guides):

* Classify data that an application has processed, transmitted, or stored. Classify sensitive data according to applicable privacy laws, business requirements, or regulatory obligations.
* Store only the data needed, then discard once the action is completed.
* Encrypt all data during transmission and at rest.
* Avoid using legacy protocols to transport sensitive data.

3. Injection

[Injection vulnerabilities](https://owasp.org/Top10/A03_2021-Injection/) can be detected through source-code review. This category includes cross-site scripting, SQL injection, and XML injection among many others. Automation can help here by making sure all parameters and [data inputs are tested to identify vulnerabilities](https://vulcan.io/blog/security-testing-101/).

Applications are vulnerable to injection when:

* User-entered data is accepted without validation, sanitization, or filtering.
* Hostile data is used to extract sensitive information.

Prevention

Keeping queries and commands separate from data is critical to preventing attempts at injection:

* Ensure that escape syntax includes special characters for the interpreter being used.
* Use query controls to prevent unexpected input from performing unauthorized actions.
* Use prepared statements with parameterized queries through a safe API, separate from the interpreter.

4. Insecure design

[Insecure design](https://owasp.org/Top10/A04_2021-Insecure_Design/) differs from insecure implementation. A secure design can be implemented imperfectly resulting in vulnerabilities. Insecure design can’t be fixed through implementation since the design itself doesn’t contain appropriate security controls. A failure to accurately assess business risk associated with the software or system under development leads to insufficient levels of security.

Prevention

A culture of security includes using a secure design methodology that evaluates threats and ensures code is designed and tested against known attack methods. Prevention includes methods that foster a secure development culture:

* Partner with application security professions to help evaluate and design controls around privacy and security. Use a secure development lifecycle.
* Components should be ready to use and design patterns that are secure.
* Apply threat modeling against access controls, key data flows, business logic, and critical authentication.

5. Security misconfiguration

[Security misconfigurations](https://owasp.org/Top10/A05_2021-Security_Misconfiguration/) can be caused by an array of inappropriately configured controls as well as other factors which contribute to application vulnerability. This category includes many common misconfigurations:

* [Misconfigured permissions for cloud](https://vulcan.io/blog/cloud-misconfiguration/) services.
* Enabling unnecessary features, which may lead to needless opened ports, services, or incorrectly elevated privileges.
* Unchanged default account login credentials.

Prevention

Prevention begins with a thorough security configuration process that is repeatable across systems and preferably automated:

* Establish a repeatable security hardening process, ideally through automation, to ensure new environments are secured appropriately with every deployment.
* Use only what is needed. Uninstall or remove unneeded features and components.
* Deploy an automated process to review security settings across environments.

[**White paper: Cloud security risks in 2022, where we stand?**](https://vulcan.io/resources/cloud-security-risks-in-2022-where-we-stand/)

6. Vulnerable and outdated components

[Unpatched and legacy components](https://owasp.org/Top10/A06_2021-Vulnerable_and_Outdated_Components/) that remain in production well after vulnerabilities are discovered and disclosed can be a major risk. Applications can be vulnerable when they aren’t running the latest software version. If it’s unclear which library or component version is being used, the application may be vulnerable. Components that aren’t scanned for vulnerabilities may also be at risk.

Prevention

Establishing a [patch-management process](https://vulcan.io/blog/patch-management-process/) can help alleviate the potential for attack by closing vulnerabilities before they become an issue. This should include:

* Removal of unused or unnecessary libraries, components, frameworks, documentation, and files.
* Continual monitoring and inventory of server-side and client-side components.
* Use of only official libraries and sources through secure links.
* Monitoring for unsupported libraries and components that are no longer maintained or have reached end of life.

7. Identification and authentication failures

[Authentication and identification failures](https://owasp.org/Top10/A07_2021-Identification_and_Authentication_Failures/) happen when user identity, authentication, and session information aren’t confirmed before the user is permitted to access systems and data. Factors that may put an application at risk due to these failures include allowing weak passwords; using weakly hashed, plain-text password data stores; and allowing bots, which can perform automated attacks such brute-force and credential stuffing.

Prevention

Prevention focuses on providing [secure password storage and retrieval](https://vulcan.io/blog/7-tips-that-will-improve-your-cyber-security-system/) and includes:

* Implementing multi-factor authentication.
* Avoiding deployment using default credentials, especially for administrative accounts.
* Limit exposure to account enumeration.

8. Software and data integrity failures

New to the OWASP list is the CWE of [failures in software and data integrity](https://owasp.org/Top10/A08_2021-Software_and_Data_Integrity_Failures/). The risk here is trusting data and software updates without checking their integrity. Attackers have used the software supply chain to issue malware through seemingly legitimate software updates. Many systems use automated software update features that do not verify the integrity of updates.

Prevention

Prevention begins with verification and includes:

* Using digital signatures or other verification methods to digitally sign software updates to ensure they’ve come from expected sources and have arrived intact.
* Verifying that third-party libraries and other dependencies originate from legitimate sources.
* Verifying that third-party resources contain no vulnerabilities by using automated security tools designed for the software supply chain.

9. Security logging and monitoring failures

The [security logging and monitoring failures](https://owasp.org/Top10/A09_2021-Security_Logging_and_Monitoring_Failures/) category focuses on issues with audit logs and monitoring during an attack. Security monitoring and logs are essential to detect and mitigate an active breach. Failures happen when:

* Logging doesn’t keep track of transactions with high value, login attempts, and failed login attempts.
* Errors and warnings generate unclear, inadequate, or no log entries.
* APIs and applications aren’t monitored for suspicious activities.
* Security logs are only available locally.
* Applications that can neither detect nor issue timely alerts for attacks in progress.

Prevention

Prevention focuses on enabling security logging and monitoring across applications. Developers should ensure security controls are implemented where appropriate. Security controls should include the following:

* Login, access control, and server-side validation failures should be logged with user context to ensure malicious and suspicious activity can be preserved long enough to allow for analysis.
* Logs should be generated in an appropriate format for log management tools to read.
* Enable monitoring and alerting for suspicious activities.
* Adopt an incident response and mitigation plan.

10. Server-side request forgery (SSRF)

The [server-side request forgery](https://owasp.org/Top10/A10_2021-Server-Side_Request_Forgery_%28SSRF%29/) category focuses on weaknesses within user-convenience features. SSRF flaws happen when web applications fetch user-requested remote sources without verifying the destination first. Specific requests can be sent to the application through the unexpected source.

Applications commonly fetch URLs to enable easier task-switching for end-users, often keeping them in the application while providing access to another feature through the fetched URL. Ever-increasing cloud architecture complexity means SSRF is occurring at a higher frequency.

Prevention

SSRF occurs at the network and application levels. Protect networks by using network segmentation to separate remote resources. Block other, nonessential traffic with “deny-by-default” policies.

Application protection methods should include:

* Data input sanitization, validation, and filtering.
* Disabling HTTP redirection at the server level
* Ensuring server responses received conform to expected results. Raw responses from the server should never be sent to the client.