LEMUEL OKECHUKWU

VEPH/20B/CY082

TASK 33A

**A Comprehensive Guide to Securing Your Web Applications**

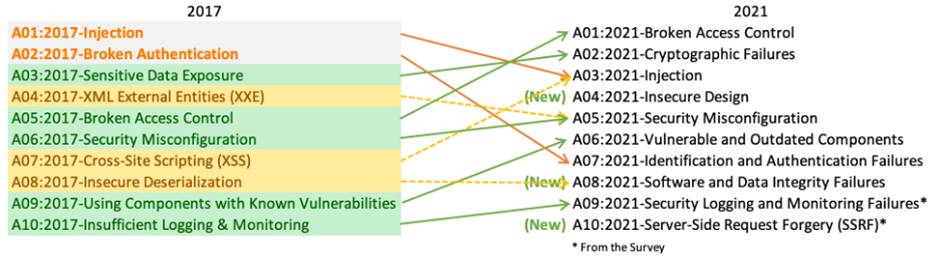
**TASK 33A:**

* Provide a comprehensive explanation of the OWASP Top 10 Web Application Security Risks, detailing how each vulnerability occurs, the underlying factors contributing to their exploitation, and the recommended security controls and best practices to mitigate these risks.
* FIRST 5

[](https://www.bytesnipers.com/en/cybersecurity-blog/owasp-top-10-web-application-security-risks)

OWASP is a nonprofit organization dedicated to improving the security of software. It provides freely available resources, tools, and guidelines to help organizations and individuals develop, deploy, and maintain secure software applications. Reference used [*OWASP Top Ten | OWASP Foundation*](https://owasp.org/www-project-top-ten/)*.*It focuses on various aspects of application security, including identifying common security risks, creating tools and documentation to address those risks, and fostering a community where knowledge and best practices can be shared.

Every few years, OWASP analyzes the top ten risks, publishes a description of these vulnerabilities and how to fix them. It is intended to raise awareness about common security issues in software development and help organizations prioritize their efforts to address these risks. The last list was compiled in 2021. Below are some of the basic information on top 10 web Application Vulnerabilities along with the comparison of last compiled 2017 OWASP top 10 list.



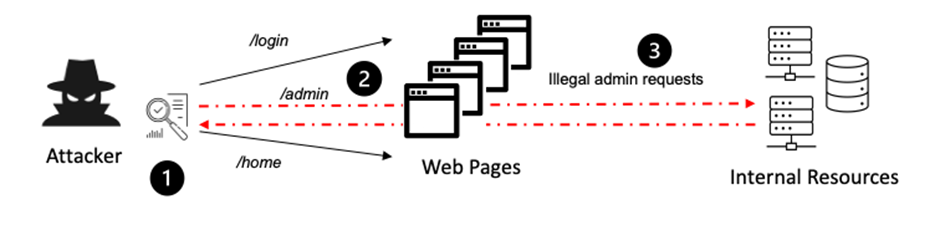
**1. BROKEN ACCESS CONTROL**

[***A01:2021-Broken Access Control***](https://owasp.org/Top10/A01_2021-Broken_Access_Control/)***moves up from the fifth position; 94% of applications were tested for some form of broken access control. The 34 Common Weakness Enumerations (CWEs) mapped to Broken Access Control had more occurrences in applications than any other category.***

Broken access control refers to vulnerabilities that occur when an application fails to properly enforce access controls, allowing unauthorized users to access resources or perform actions they should not have permission for. Access control is about restricting access to resources based on user roles and permissions.

Two Extends of this Vulnerability:

1. **API Endpoint Manipulation:** For example, consider the API endpoint admin.example.com/messages/view=article&id=126&Itemid=319. By altering parameters, an attacker could request data for a different article (e.g., id=127), breaching the intended access control.
2. **URL-Based Exploitation**: For example, In the context of URLs, example.com/app/get.info is a permissible request for general information. However, an unauthorized attempt to access an admin page (example.com/app/admin\_get.info) can result in a violation of access control.



Just by changing the URL, attacker can access internal resources.

***Common Issues in Broken Access Control:***

1. Lack of proper authorization checks that allow users to access restricted functionalities.
2. Insufficient validation of user input when performing sensitive actions.
3. Misconfigured access controls that grant users excessive privileges.

**Strategies for Mitigation:**

1. Grant users the minimum access required for their functionalities.
2. Assign distinct identifiers to users, preventing malicious actors from manipulating references to gain unauthorized access to sensitive resources.
3. Maintain detailed logs of access control failures.
4. Implement rate-limiting mechanisms to prevent automated attacks. Common example is after 5 fail attempts you have to wait for 10 mintues minimum.
5. Develop and maintain a comprehensive emergency response plan.

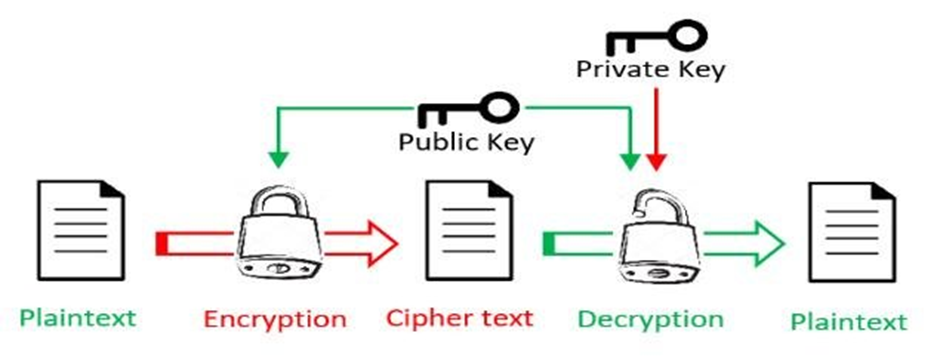
**2. CRYPTOGRAPHIC FAILURES**

[***A02:2021-Cryptographic Failures***](https://owasp.org/Top10/A02_2021-Cryptographic_Failures/)***shifts up one position to #2, previously known as Sensitive Data Exposure, which was broad symptom rather than a root cause. The renewed focus here is on failures related to cryptography which often leads to sensitive data exposure or system compromise.***

It refers to security vulnerabilities that arise due to weaknesses or improper implementation of cryptographic mechanisms.

Cryptographic failures pose the risk of exposing sensitive data, leaving information such as passwords, credit card numbers, health data, and GDPR-protected information vulnerable to unauthorized access. Weak or outdated cryptographic algorithms and inadequate key management practices contribute to this susceptibility.

In the context of data transmission, the choice between HTTP and HTTPS is important for safeguarding sensitive information. While HTTP exposes data to interception due to the lack of encryption whereas HTTPS uses SSL/TLS protocols, ensures secure and encrypted data exchange. However, just the presence of an SSL certificate doesn’t guarantee immunity. Example: A website supports an outdated version of the SSL/TLS protocol (e.g., SSL 2.0 or TLS 1.0), which is known to have security vulnerabilities. Attackers can exploit these vulnerabilities to perform attacks like man-in-the-middle attacks or downgrade attacks.



Working of Asymmetric Cryptography

**Common Issues:**

1. Use of deprecated or outdated or weak cryptographic algorithms exposes data to vulnerabilities.
2. Inadequate management of cryptographic keys, including weak key generation or storage practices, compromises data security.
3. Transmitting sensitive data over unsecured HTTP channels exposes it to interception and unauthorized access.

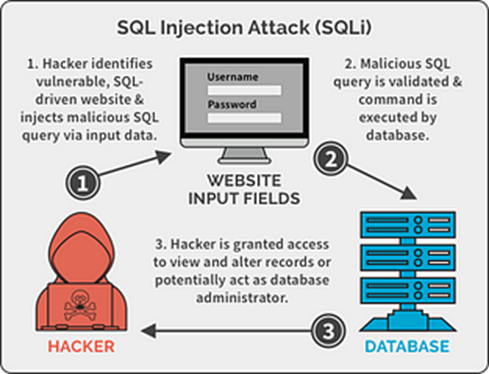
**Strategies for Mitigation:**

1. Utilize up-to-date and robust cryptographic algorithms to ensure the security of sensitive information.
2. Implement secure key generation, storage, and distribution practices to fortify the encryption foundation.
3. Mandate the use of HTTPS to encrypt data during transmission, preventing unauthorized interception.
4. Conduct routine audits to identify and rectify vulnerabilities in cryptographic implementations and key management practices.
5. Regularly update systems and protocols to ensure compliance with the latest security standards.

**3. INJECTION**

[***A03:2021-Injection***](https://owasp.org/Top10/A03_2021-Injection/)***slides down to the third position. 94% of the applications were tested for some form of injection, and the 33 CWEs mapped into this category have the second most occurrences in applications. Cross-site Scripting is now part of this category in this edition.***

Injection vulnerability is a grave concern, not confined to SQL but extending to variants like Cross-Site Scripting (XSS). Imagine a scenario where user input isn’t properly checked or cleaned by an application — that’s the breeding ground for injection attacks. This vulnerability surfaces when dynamic queries or non-parameterized calls are sent straight to interpreters without context or proper precautions. It’s like leaving a door wide open for malicious commands, such as SQL injection, enabling attackers to manipulate queries and execute unintended actions with potential consequences. In simpler terms, when an application doesn’t thoroughly check what users input, it becomes vulnerable to sneaky attacks that can mess with its core functions.



**Common Issues:**

1. Lack of Input Validation means neglecting to check and validate what users input into an application can be risky. It’s like allowing anything to go in without making sure it’s safe.
2. If the application doesn’t clean up or sanitize (insufficient sanitization) the data it receives, it’s like letting in unfiltered content. This could include harmful commands or scripts that might do things they shouldn’t, like gaining unauthorized access or performing actions without permission.
3. Dynamic Query Execution Without Parameters. Imagine running commands without checking them properly. When an application executes dynamic queries without verifying the instructions. It might accidentally follow harmful commands injected by someone with bad intentions.

**Strategies for Mitigation:**

1. Implement parameterized queries to ensure that user inputs are treated as data rather than executable code, preventing injection attacks.
2. Enforce strict input validation and sanitization practices to filter out potentially malicious characters or commands, reducing the risk of injection vulnerabilities.
3. Apply context-aware escaping methods to ensure that user-supplied data is interpreted correctly, mitigating the risk of unintended code execution.
4. Employ prepared statements in database queries to separate SQL code from user input, preventing attackers from manipulating the query structure.
5. Educate developers on secure coding practices and conduct regular code reviews to identify and rectify potential injection vulnerabilities in the application codebase.

**4. INSECURE DESIGN**

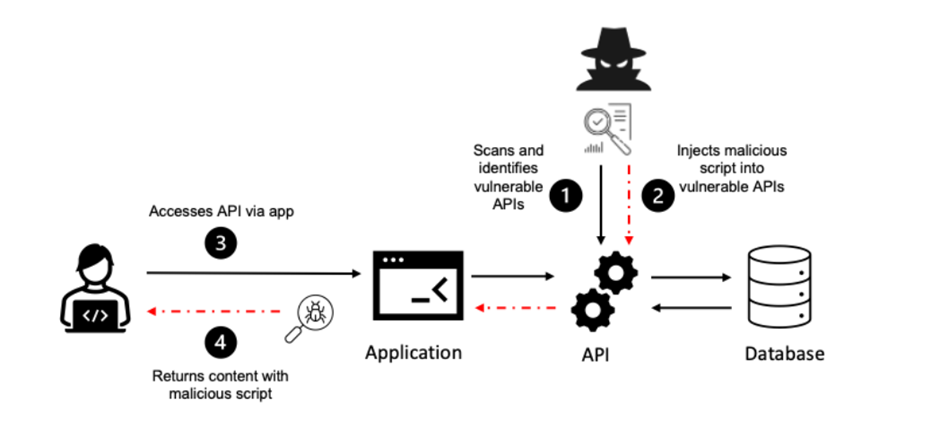
[***A04:2021-Insecure Design***](https://owasp.org/Top10/A04_2021-Insecure_Design/)***is a new category for 2021, with a focus on risks related to design flaws. If we genuinely want to “move left” as an industry, it calls for more use of threat modeling, secure design patterns and principles, and reference architectures.***

Insecure design refers to vulnerabilities that arise due to flawed or inadequate design decisions in the architecture of a software application.

These encompass various weaknesses such as ineffective control design and insecure implementation. It emphasizes the industry’s need to shift focus leftward, advocating for increased utilization of threat modeling, secure design patterns, principles, and reference architectures.

Secure design doesn’t totally guarantee secure implementation. In some cases, a flaw in the design phase may persist despite a secure implementation effort.

Secure design involves a proactive methodology, continually assessing threats to ensure that the code is robustly conceived and rigorously tested. It places a premium on preventive measures over reactive fixes.



**Common Issues:**

1. Flaws in designing controls that should ensure security, leading to vulnerabilities.
2. Challenges arise when the implementation fails to align with the secure design, resulting in exploitable weaknesses.
3. Instances where the design and implementation phases are misaligned, allowing vulnerabilities to persist.

**Strategies for Mitigation:**

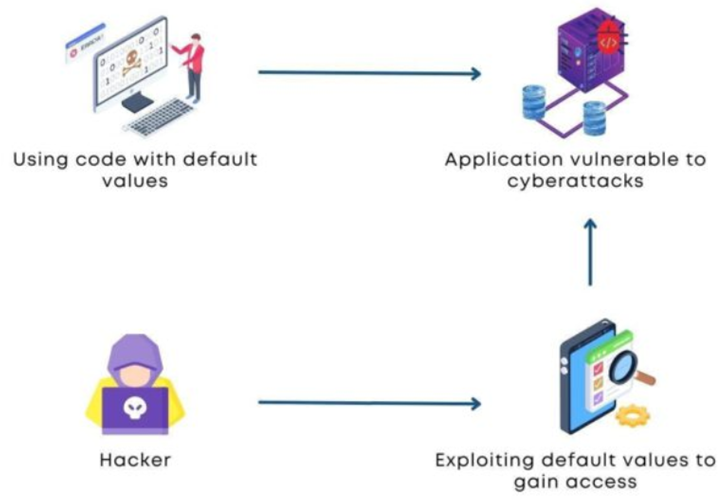
1. Integrate security into every phase of the development life cycle, ensuring that security considerations are systematically addressed.
2. Utilize tools and frameworks that prioritize security, helping developers produce more secure code with fewer vulnerabilities.
3. Adopt threat modeling practices to identify potential threats, considering both intended (use cases) and unintended (misuse cases) system interactions.
4. Implement thorough code reviews with a security focus to identify and rectify design-related vulnerabilities before they manifest.
5. Provide ongoing training to developers on secure design principles and best practices, empowering them to embed security into their design decisions.

**5. SECURITY MISCONFIGURATION**

[***A05:2021-Security Misconfiguration***](https://owasp.org/Top10/A05_2021-Security_Misconfiguration/)***moves up from #6 in the previous edition; 90% of applications were tested for some form of misconfiguration. With more shifts into highly configurable software, it’s not surprising to see this category move up. The former category for XML External Entities (XXE) is now part of this category.***

Occur when an application, server, or system is not set up properly, leaving it vulnerable to exploitation. This category, which now includes the former XML External Entities (XXE) issues, highlights instances where security components within applications are present but improperly configured. XXE is because default or improper configurations may allow an attacker to exploit the application by injecting malicious XML content. This could lead to sensitive information disclosure, denial of service, or other forms of attacks.

Consider a scenario where a user encounters a detailed error message explaining why an error occurred and what actions to take. While this may seem helpful, it inadvertently exposes the application to potential exploitation. Such detailed error messages, coupled with excessive information, fall under security misconfiguration, providing attackers insights they can leverage.



Having a default password comes security misconfiguration.

**Common Issues:**

1. Security is compromised when non-essential features are unnecessarily enabled or installed, providing potential entry points for exploitation.
2. Not Changing change default accounts and passwords poses a significant risk, as attackers can exploit these unchanged credentials.
3. Issues arise when security configurations present in one version are disabled or downgraded in subsequent updates, leaving the application vulnerable.

**Strategies for Mitigation:**

1. Implement hardening guides that provide a set of security best practices for configuring systems and applications to minimize vulnerabilities.
2. Remove unnecessary features and components from the application to minimize the attack surface and potential misconfigurations.
3. Send security directives to clients via headers, especially in HTTPS configurations, to enhance security measures and control client behavior.
4. Implement automation tools to ensure consistent and secure configurations, reducing the risk of human errors in manual setups.
5. Conduct routine security audits to proactively identify and rectify misconfigurations, ensuring ongoing alignment with security best practices.