***ASSIGNMENT-1***

***Owasp top 5 vulnerabilities***

**

***A01:2021 – Broken Access Control***

*Access control enforces policy such that users cannot act outside of their intended permissions. Failures typically lead to unauthorized information disclosure, modification, or destruction of all data or performing a business function outside the user's limits. Common access control vulnerabilities include:*

*Violation of the principle of least privilege allows access to certain capabilities, roles, or users without permission. This can be achieved through URL modification, internal application state, HTML page modification, or API requests. Permission to view or edit someone else's account, accessing APIs with missing access controls, and escalating privileges can also be exploited. Metadata manipulation and CORS misconfiguration can also lead to API access from unauthorized sources.*

*Access control is only effective in trusted server-side code or server-less API, where the attacker cannot modify the access control check or metadata.*

* *Except for public resources, deny by default.*
* *Implement access control mechanisms once and re-use them throughout the application, including minimizing Cross-Origin Resource Sharing (CORS) usage.*
* *Model access controls should enforce record ownership rather than accepting that the user can create, read, update, or delete any record.*
* *Unique application business limit requirements should be enforced by domain models.*
* *Disable web server directory listing and ensure file metadata (e.g., .git) and backup files are not present within web roots.*

## ***Example Attack Scenarios***

***Scenario #1:****The application uses unverified data in a SQL call that is accessing account information:*

*pstmt.setString(1, request.getParameter("acct"));*

*ResultSet results = pstmt.executeQuery( );*

*An attacker simply modifies the browser's 'acct' parameter to send whatever account number they want. If not correctly verified, the attacker can access any user's account.*

*https://example.com/app/accountInfo?acct=notmyacct*

|  |
| --- |
| *CWE-22 Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')* |
|  |
| *CWE-23 Relative Path Traversal* |
|  |
| *CWE-35 Path Traversal: '.../...//'* |
|  |
| *CWE-59 Improper Link Resolution Before File Access ('Link Following')* |
|  |
| *CWE-200 Exposure of Sensitive Information to an Unauthorized Actor* |
|  |
| *CWE-201 Exposure of Sensitive Information Through Sent Data* |
|  |
| *CWE-219 Storage of File with Sensitive Data Under Web Root* |
|  |
| *CWE-264 Permissions, Privileges, and Access Controls (should no longer be used)* |
|  |
| *CWE-275 Permission Issues* |
|  |
| *CWE-276 Incorrect Default Permissions* |
|  |
| *CWE-284 Improper Access Control* |
|  |
| *CWE-285 Improper Authorization* |
|  |
| *CWE-352 Cross-Site Request Forgery (CSRF)* |
|  |
| *CWE-359 Exposure of Private Personal Information to an Unauthorized Actor* |
|  |
| *CWE-377 Insecure Temporary File* |
|  |
| *CWE-402 Transmission of Private Resources into a New Sphere ('Resource Leak')* |
|  |
| *CWE-425 Direct Request ('Forced Browsing')* |
|  |
| *CWE-441 Unintended Proxy or Intermediary ('Confused Deputy')* |
|  |
| *CWE-497 Exposure of Sensitive System Information to an Unauthorized Control Sphere* |
|  |
| *CWE-538 Insertion of Sensitive Information into Externally-Accessible File or Directory* |
|  |
| *CWE-540 Inclusion of Sensitive Information in Source Code* |
|  |
| *CWE-548 Exposure of Information Through Directory Listing* |
|  |
| *CWE-552 Files or Directories Accessible to External Parties* |
|  |
| *CWE-566 Authorization Bypass Through User-Controlled SQL Primary Key* |
|  |
| *CWE-601 URL Redirection to Untrusted Site ('Open Redirect')* |

# *A02:2021 – Cryptographic Failures*

*Cryptographic Failures, as listed in the OWASP Top Ten Project, encompass a range of vulnerabilities and weaknesses in the implementation of cryptographic mechanisms within software applications. These weaknesses can lead to Description:*

*Cryptographic Failures, as listed in the OWASP Top Ten Project, encompass a range of vulnerabilities and weaknesses in the implementation of cryptographic mechanisms within software applications. These weaknesses can lead to security breaches, data leaks, and unauthorized access. Examples include the use of deprecated or weak cryptographic algorithms, improper key management, insufficient entropy for random number generation, and insecure handling of cryptographic operations.*

*Prevention:*

*To mitigate Cryptographic Failures, adhere to the following best practices:*

*Strong Algorithms and Key Lengths: Always use modern, strong cryptographic algorithms and key lengths. Avoid deprecated algorithms like MD5 or weak keys that could be easily exploited.*

*Proper Key Management: Implement secure key management practices, including generating and storing cryptographic keys in a protected environment. Use key derivation functions for password-based keys.*

*Secure Initialization Vectors (IVs): Generate unique and random IVs for encryption operations to prevent patterns and improve security.*

*Use Authenticated Encryption: Employ authenticated encryption modes like AES-GCM or AES-CCM instead of plain encryption to ensure both confidentiality and integrity.*

*Certificate Validation: Validate server certificates and their trust chains to prevent man-in-the-middle attacks.security breaches, data leaks, and unauthorized access. Examples include the use of deprecated or weak cryptographic algorithms, improper key management, insufficient entropy for random number generation, and insecure handling of cryptographic operations.*

*Prevention: To mitigate Cryptographic Failures, adhere to the following best practices:*

*1. Strong Algorithms and Key Lengths: Always use modern, strong cryptographic algorithms and key lengths. Avoid deprecated algorithms like MD5 or weak keys that could be easily exploited.*

*2. Proper Key Management: Implement secure key management practices, including generating and storing cryptographic keys in a protected environment. Use key derivation functions for password-based keys.*

*3. Secure Initialization Vectors (IVs): Generate unique and random IVs for encryption operations to prevent patterns and improve security.*

*4. Use Authenticated Encryption: Employ authenticated encryption modes like AES-GCM or AES-CCM instead of plain encryption to ensure both confidentiality and integrity.*

*5. Certificate Validation: Validate server certificates and their trust chains to prevent man-in-the-middle attacks.*

## ***Example Attack Scenarios***

***Scenario #1****: An application encrypts credit card numbers in a database using automatic database encryption. However, this data is automatically decrypted when retrieved, allowing a SQL injection flaw to retrieve credit card numbers in clear text.*

|  |
| --- |
| *CWE-261 Weak Encoding for Password* |
|  |
| *CWE-296 Improper Following of a Certificate's Chain of Trust* |
|  |
| *CWE-310 Cryptographic Issues* |
|  |
| *CWE-319 Cleartext Transmission of Sensitive Information* |
|  |
| *CWE-321 Use of Hard-coded Cryptographic Key* |
|  |
| *CWE-322 Key Exchange without Entity Authentication* |
|  |
| *CWE-323 Reusing a Nonce, Key Pair in Encryption* |
|  |
| *CWE-324 Use of a Key Past its Expiration Date* |
|  |
| *CWE-325 Missing Required Cryptographic Step* |
|  |
| *CWE-326 Inadequate Encryption Strength* |
|  |
| *CWE-327 Use of a Broken or Risky Cryptographic Algorithm* |
|  |
| *CWE-328 Reversible One-Way Hash* |
|  |
| *CWE-329 Not Using a Random IV with CBC Mode* |
|  |
| *CWE-330 Use of Insufficiently Random Values* |
|  |
| *CWE-331 Insufficient Entropy* |
|  |
| *CWE-335 Incorrect Usage of Seeds in Pseudo-Random Number Generator(PRNG)* |
|  |
| *CWE-336 Same Seed in Pseudo-Random Number Generator (PRNG)* |
|  |
| *CWE-337 Predictable Seed in Pseudo-Random Number Generator (PRNG)* |
|  |
| *CWE-338 Use of Cryptographically Weak Pseudo-Random Number Generator(PRNG)* |
|  |
| *CWE-340 Generation of Predictable Numbers or Identifiers* |
|  |
| *CWE-347 Improper Verification of Cryptographic Signature* |

# *A03:2021 – Injection*

*Injection, listed in the OWASP Top Ten Project, refers to vulnerabilities that arise when untrusted data is improperly handled by an application and then executed as part of a command or query. Common types include SQL injection, where malicious SQL commands are inserted into input fields, and OS command injection, where arbitrary commands are executed on the underlying operating system. Injection attacks can lead to data leaks, unauthorized access, and even complete system compromise.*

***Prevention:*** *To prevent Injection attacks, follow these best practices:*

1. ***Input Validation:*** *Validate and sanitize all user inputs. Use whitelisting to allow only expected values and characters, rejecting any unexpected input.*
2. ***Parameterized Queries:*** *Use parameterized queries (prepared statements) or ORM frameworks to separate data from SQL commands, preventing SQL injection attacks.*
3. ***Stored Procedures:*** *Utilize stored procedures and prepared statements in databases to prevent direct execution of malicious input.*
4. ***Context-Specific Escaping:*** *Apply context-specific escaping for output data. Use libraries that automatically escape output based on the context it will be used in (e.g., HTML, JavaScript).*
5. ***Least Privilege:*** *Ensure that database accounts, operating system accounts, and application components have the minimum necessary privileges.*

## ***Example Attack Scenarios***

***Scenario #1:****An application uses untrusted data in the construction of the following vulnerable SQL call:*

*String query = "SELECT \\* FROM accounts WHERE custID='" + request.getParameter("id")*

|  |
| --- |
| *CWE-20 Improper Input Validation* |
|  |
| *CWE-74 Improper Neutralization of Special Elements in Output Used by a Downstream Component ('Injection')* |
|  |
| *CWE-75 Failure to Sanitize Special Elements into a Different Plane (Special Element Injection)* |
|  |
| *CWE-77 Improper Neutralization of Special Elements used in a Command ('Command Injection')* |
|  |
| *CWE-78 Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection')* |
|  |
| *CWE-79 Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')* |
|  |
| *CWE-80 Improper Neutralization of Script-Related HTML Tags in a Web Page (Basic XSS)* |
|  |
| *CWE-83 Improper Neutralization of Script in Attributes in a Web Page* |
|  |
| *CWE-87 Improper Neutralization of Alternate XSS Syntax* |
|  |
| *CWE-88 Improper Neutralization of Argument Delimiters in a Command ('Argument Injection')* |
|  |
| *CWE-89 Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')* |
|  |
| *CWE-90 Improper Neutralization of Special Elements used in an LDAP Query ('LDAP Injection')* |
|  |
| *CWE-91 XML Injection (aka Blind XPath Injection)* |
|  |
| *CWE-93 Improper Neutralization of CRLF Sequences ('CRLF Injection')* |
|  |
| *CWE-94 Improper Control of Generation of Code ('Code Injection')* |
|  |
| *CWE-95 Improper Neutralization of Directives in Dynamically Evaluated Code ('Eval Injection')* |
|  |
| *CWE-96 Improper Neutralization of Directives in Statically Saved Code ('Static Code Injection')* |
|  |
| *CWE-97 Improper Neutralization of Server-Side Includes (SSI) Within a Web Page* |
|  |
| *CWE-98 Improper Control of Filename for Include/Require Statement in PHP Program ('PHP Remote File Inclusion')* |
|  |
| *CWE-99 Improper Control of Resource Identifiers ('Resource Injection')* |
|  |
| *CWE-100 Deprecated: Was catch-all for input validation issues* |
|  |
| *CWE-113 Improper Neutralization of CRLF Sequences in HTTP Headers ('HTTP Response Splitting')* |
|  |
| *CWE-116 Improper Encoding or Escaping of Output* |
|  |
| *CWE-138 Improper Neutralization of Special Elements* |
|  |
| *CWE-184 Incomplete List of Disallowed Inputs* |
|  |
| *CWE-470 Use of Externally-Controlled Input to Select Classes or Code ('Unsafe Reflection')* |
|  |
| *CWE-471 Modification of Assumed-Immutable Data (MAID)* |

# *A04:2021 – Insecure Design*

*Insecure Design, a category outlined in the OWASP Top Ten Project, pertains to vulnerabilities stemming from poor design decisions in the architecture of software applications. This includes flaws in how security mechanisms, authentication processes, access controls, and other security features are implemented. Insecure design can result in exploitable vulnerabilities that attackers can leverage to compromise the application's integrity and expose sensitive data.*

### ***Requirements and Resource Management***

*Collect and negotiate the business requirements for an application with the business, including the protection requirements concerning confidentiality, integrity, availability, and authenticity of all data assets and the expected business logic. Take into account how exposed your application will be and if you need segregation of tenants (additionally to access control). Compile the technical requirements, including functional and non-functional security requirements. Plan and negotiate the budget covering all design, build, testing, and operation, including security activities.*

***Prevention:*** *To mitigate Insecure Design vulnerabilities, adhere to the following best practices*

*:*

1. ***Threat Modeling:*** *Conduct comprehensive threat modeling during the design phase to identify potential security risks and address them proactively.*
2. ***Secure Architecture Patterns:*** *Utilize well-established secure architecture patterns and principles that are known to mitigate common vulnerabilities.*
3. ***Least Privilege:*** *Implement the principle of least privilege, granting users and components only the necessary access rights and permissions.*
4. ***Strong Authentication:*** *Implement strong and multi-factor authentication mechanisms to ensure that only authorized users can access the application.*
5. ***Authorization and Access Controls:*** *Enforce proper authorization and access controls based on user roles and responsibilities.*

*Example Attack Scenarios*

***Scenario #1:****A credential recovery workflow might include “questions and answers,” which is prohibited by NIST 800-63b, the OWASP ASVS, and the OWASP Top 10. Questions and answers cannot be trusted as evidence of identity as more than one person can know the answers, which is why they are prohibited. Such code should be removed and replaced with a more secure design.*

|  |
| --- |
| *CWE-73 External Control of File Name or Path* |
|  |
| *CWE-183 Permissive List of Allowed Inputs* |
|  |
| *CWE-209 Generation of Error Message Containing Sensitive Information* |
|  |
| *CWE-213 Exposure of Sensitive Information Due to Incompatible Policies* |
|  |
| *CWE-235 Improper Handling of Extra Parameters* |
|  |
| *CWE-256 Unprotected Storage of Credentials* |
|  |
| *CWE-257 Storing Passwords in a Recoverable Format* |
|  |
| *CWE-266 Incorrect Privilege Assignment* |
|  |
| *CWE-269 Improper Privilege Management* |
|  |
| *CWE-280 Improper Handling of Insufficient Permissions or Privileges* |
|  |
| *CWE-311 Missing Encryption of Sensitive Data* |
|  |
| *CWE-312 Cleartext Storage of Sensitive Information* |
|  |
| *CWE-313 Cleartext Storage in a File or on Disk* |
|  |
| *CWE-316 Cleartext Storage of Sensitive Information in Memory* |
|  |
| *CWE-419 Unprotected Primary Channel* |
|  |
| *CWE-430 Deployment of Wrong Handler* |
|  |
| *CWE-434 Unrestricted Upload of File with Dangerous Type* |
|  |
| *CWE-444 Inconsistent Interpretation of HTTP Requests ('HTTP Request Smuggling')* |
|  |
| *CWE-451 User Interface (UI) Misrepresentation of Critical Information* |
|  |
| *CWE-472 External Control of Assumed-Immutable Web Parameter* |
|  |
| *CWE-501 Trust Boundary Violation* |
|  |
| *CWE-522 Insufficiently Protected Credentials* |
|  |
| *CWE-525 Use of Web Browser Cache Containing Sensitive Information* |

# *A05:2021 – Security Misconfiguration*

*Security Misconfiguration, part of the OWASP Top Ten Project, refers to vulnerabilities that arise from improper configuration of security settings in software applications, databases, servers, and other components. These misconfigurations can lead to unauthorized access, data leaks, and even system compromise. Common examples include default credentials, exposed sensitive information, and unnecessary open ports.*

***Prevention:*** *To prevent Security Misconfigurations, follow these best practices:*

1. ***Secure Defaults:*** *Set secure default configurations for all components, services, and frameworks used in the application.*
2. ***Least Privilege:*** *Apply the principle of least privilege to limit access rights and permissions to what is necessary for each user or component.*
3. ***Regular Updates and Patches:*** *Keep all software, libraries, and frameworks up to date to mitigate known vulnerabilities.*
4. ***Access Controls:*** *Implement proper access controls based on user roles and responsibilities to prevent unauthorized access.*
5. ***Default Credentials:*** *Change default credentials for all components, including databases, APIs, and administrative interfaces.*
6. ***Error Handling:*** *Ensure that error messages do not reveal sensitive information and provide only essential information to users.*
7. ***Secure Headers:*** *Configure security-related HTTP headers to protect against common web vulnerabilities.*

## ***Example Attack Scenarios***

***Scenario #1:****The application server comes with sample applications not removed from the production server. These sample applications have known security flaws attackers use to compromise the server. Suppose one of these applications is the admin console, and default accounts weren't changed. In that case, the attacker logs in with default passwords and takes over.*

|  |
| --- |
| *CWE-2 7PK - Environment* |
| *CWE-11 ASP.NET Misconfiguration: Creating Debug Binary* |
| *CWE-13 ASP.NET Misconfiguration: Password in Configuration File* |
| *CWE-15 External Control of System or Configuration Setting* |
| *CWE-16 Configuration* |
| *CWE-260 Password in Configuration File* |
| *CWE-315 Cleartext Storage of Sensitive Information in a Cookie* |
| *CWE-520 .NET Misconfiguration: Use of Impersonation* |
| *CWE-526 Exposure of Sensitive Information Through Environmental Variables* |
| *CWE-537 Java Runtime Error Message Containing Sensitive Information* |
| *CWE-541 Inclusion of Sensitive Information in an Include File* |
| *CWE-547 Use of Hard-coded, Security-relevant Constants* |
| *CWE-611 Improper Restriction of XML External Entity Reference* |
| *CWE-614 Sensitive Cookie in HTTPS Session Without 'Secure' Attribute* |
| *CWE-756 Missing Custom Error Page* |
| *CWE-776 Improper Restriction of Recursive Entity References in DTDs ('XML Entity Expansion')* |
| *CWE-942 Permissive Cross-domain Policy with Untrusted Domains* |
| *CWE-1004 Sensitive Cookie Without 'HttpOnly' Flag* |
| *CWE-1032 OWASP Top Ten 2017 Category A6 - Security Misconfiguration* |
| *CWE-1174 ASP.NET Misconfiguration: Improper Model Validation* |
|  |