TOP 100 Vulnerabilities Guide Handbook

Injection Vulnerabilities

SQL Injection (SQLi)

Parameters:

- User inputs not properly sanitized before inclusion in SQL queries.
- Dynamic generation of SQL queries in code.
- Use of concatenated strings to construct SQL queries.
- Lack of input validation for SQL queries.
- Find any User input which retrieve data or communicate from database!

Steps to follow:

1. Identify user input fields in web forms or URL parameters.
2. Input malicious SQL commands into these fields.
3. Observe for any SQL errors or unexpected behaviour.
4. If errors or unexpected behaviour occur, it indicates vulnerability to SQL injection.
5. Try and test different SQL injection techniques such as UNION-based, Boolean-based, or time-based to exploit the vulnerability.

Cross-Site Scripting (XSS)

Parameters:

- Lack of input validation and sanitization.
- Reflection of user inputs back to the web page without encoding.
- Inadequate use of Content Security Policy (CSP)
- Absence of proper output encoding in web applications
- Use of client-side scripting languages like JavaScript without proper filtering.

Steps to follow:

1. Identify user input fields or areas where user data is displayed on the web page.
2. Input scripts or JavaScript code into these fields.
3. Submit the input and observe if the script executes.
4. If the script executes, it indicates vulnerability to XSS.
5. Test for different XSS payloads such as <script>alert('XSS')</script> or <img src="javascript:alert('XSS')"> to exploit the vulnerability

One can use Burp Intruder to try multiple payloads and observe the response to exploit.

Cross-Site Request Forgery (CSRF)

Parameters:

- Absence of anti-CSRF tokens in sensitive actions.
- Lack of validation for the origin of requests.
- Vulnerable to requests that execute unauthorized actions without user consent.
- Failure to implement SameSite attribute for cookies.

Steps to follow:

1. Identify sensitive actions such as changing passwords or transferring funds.
2. Submit a request to perform these actions from an external site.
3. Observe if the action is executed without user consent.
4. If actions are executed without consent, it indicates vulnerability to CSRF.
5. Implement anti-CSRF tokens and SameSite attribute for cookies to prevent CSRF attacks.

Remote Code Execution (RCE)

Parameters:

- Vulnerable code execution environments.
- Insecure deserialization of data.
- Lack of input validation and sanitization.
- Insufficient security measures in network protocols or services.
- Execution of arbitrary code by attackers.

Steps to Follow:

1. Identify input fields or parameters that can execute code on the server like user input fields, HTTP req parameters, form data, file uploads, API endpoints, DB quries and server side scripting languages.
2. Input payloads or commands that execute code into these fields.
3. Submit the input and observe if the code is executed.
4. If code execution occurs, it indicates vulnerability to RCE.
5. Test for different RCE payloads such as command injection or shell injection to exploit the vulnerability.

Command Injection

Parameters:

- Lack of input validation and sanitization for system commands.
- Use of concatenated strings to construct system commands.
- Execution of arbitrary commands by attackers.
- Vulnerable code execution environments.

Steps to Follow:

1. Identify input fields or parameters that are used to execute system commands.
2. Input payloads or commands that execute arbitrary commands into these fields.
3. Submit the input and observe if the command is executed.
4. If command execution occurs, it indicates vulnerability to command injection.
5. Test for different command injection payloads.

XML Injection

Parameters:

- Lack of input validation and sanitization for XML data.
- Use of untrusted XML data in XML parsers.
- Execution of malicious XML code by attackers.
- Vulnerable XML processing libraries or frameworks.

Steps to follow:

1. Identify input fields or parameters that accept XML data.
2. Input payloads or XML code that can manipulate XML parsers.
3. Submit the input and observe if the code is executed or parsed.
4. If code execution or parsing occurs, it indicates vulnerability to XML injection.
5. Test for different XML injection payloads such as <![CDATA[<payload>]]> or <!ENTITY % xx '<payload>'> to exploit the vulnerability.

LDAP Injection

Parameters:

- Lack of input validation and sanitization for LDAP queries.
- Use of untrusted input in LDAP queries.
- Execution of malicious LDAP statements by attackers.
- Vulnerable LDAP libraries or frameworks.

Steps to Follow:

1. Identify input fields or parameters that accept LDAP queries.
2. Input payloads or LDAP statements that can manipulate LDAP queries.
3. Submit the input and observe if the query is executed.
4. If query execution occurs, it indicates vulnerability to LDAP injection.
5. Test for different LDAP injection payloads such as *()

XPath Injection

Parameters:

- Lack of input validation and sanitization for XPath queries.
- Use of untrusted input in XPath queries.
- Execution of malicious XPath statements by attackers.
- Vulnerable XPath processing libraries or frameworks.

Steps to Follow:

1. Identify input fields or parameters that accept XPath queries.
2. Input payloads or XPath statements that can manipulate XPath queries.
3. Submit the input and observe if the query is executed.
4. If query execution occurs, it indicates vulnerability to XPath injection.
5. Test for different XPath injection payloads such as ' or 1=1 or '1'='1' to exploit the vulnerability.

HTML Injection

Parameters:

- Lack of input validation and sanitization for HTML data.
- Reflection of untrusted HTML data back to the web page without encoding.
- Execution of malicious HTML code.
- Vulnerable HTML rendering engines or frameworks.

Steps to Follow:

1. Identify input fields or parameters that accept HTML data.
2. Input payloads or HTML code that can manipulate HTML rendering.
3. Submit the input and observe if the code is executed or rendered.
4. If code execution or rendering occurs, it indicates vulnerability to HTML injection.
5. Test for different HTML injection payloads such as "<h>malicious link</h>" to exploit the vulnerability.

Server-Side Includes (SSI) Injection

Parameters:

- Lack of input validation and sanitization for SSI commands.
- Use of untrusted input in SSI commands.
- Execution of malicious SSI code.
- Vulnerable SSI processing engines or frameworks.

Steps to Follow:

1. Identify input fields or parameters that accept SSI commands.
2. Input payloads or SSI code that can manipulate SSI processing.
3. Submit the input and observe if the code is executed.
4. If code execution occurs, it indicates vulnerability to SSI injection.
5. Test for different SSI injection payloads to exploit the vulnerability.

Implement input validation and sanitize SSI commands to mitigate the risk of SSI injection.

Broken Authentication and Session Management

Session Fixation

Parameters:

- Lack of session regeneration after authentication.
- Session IDs transmitted over unencrypted channels.
- Absence of session validation or expiration controls.

Steps to Follow:

1. Observe if session IDs remain unchanged after authentication.
2. Check if session IDs are transmitted over insecure channels.
3. Test if sessions remain active indefinitely or lack expiration controls.
4. If any of these conditions are met, it indicates vulnerability to session fixation.
5. Attempt to fixate a session by forcing a specific session ID onto a user.

Brute Force Attack

Parameters:

- Absence of account lockout mechanisms.
- Weak or predictable passwords.
- Lack of rate-limiting on login attempts.
- No monitoring for abnormal login patterns.

Steps to Follow:

1. Attempt to login to an account using various combinations of usernames and passwords.
2. Test for the absence of account lockout mechanisms or rate-limiting.
3. Monitor login attempts for any abnormal patterns.
4. If successful logins are achieved through excessive attempts, it indicates vulnerability to brute force attacks.
5. Implement strong password policies and account lockout mechanisms to mitigate the risk.

Session Hijacking

Parameters:

- Weak session management practices.
- Use of predictable session IDs.
- Lack of secure transmission protocols.
- Absence of mechanisms to detect unauthorized access.

Steps to Follow:

1. Monitor network traffic for session IDs transmitted in plain text.
2. Check for session IDs that are easily guessable or predictable.
3. Test for sessions that can be easily hijacked without detection.
4. If unauthorized access to sessions is possible, it indicates vulnerability to session hijacking.
5. Implement secure session management practices and use encryption for transmitting session data.

Password Cracking

Parameters:

- Storage of passwords in plain text or weakly hashed formats.
- Use of weak hashing algorithms.
- Lack of salting in password storage.
- No policy for password complexity or expiration.

Steps to Follow:

1. Obtain access to the password database and analyse how passwords are stored.
2. Test the strength of passwords by attempting to crack them using common techniques.
3. Check for the absence of salt in password hashing.
4. If passwords are easily cracked or stored in insecure formats, it indicates vulnerability to password cracking.
5. Implement strong password hashing algorithms, salting, and enforce password policies for better security.

Weak Password Storage

Parameters:

- Storage of passwords in plain text.
- Use of weak hashing algorithms without salting.
- Vulnerable password storage mechanisms.

Steps to Follow:

1. Review how passwords are stored in the system.
2. Check if passwords are stored in plain text or using weak hashing algorithms.
3. Test for the absence of salt in password hashing.
4. If passwords are stored insecurely, it indicates vulnerability to weak password storage.
5. Implement secure password storage mechanisms such as strong hashing algorithms and salting.

Insecure Authentication

Parameters:

- Lack of multi-factor authentication.
- Use of insecure authentication protocols such as HTTP Basic Auth.
- Absence of secure transmission channels for authentication data.

Steps to Follow:

1. Review the authentication process and protocols used.
2. Test for the absence of multi-factor authentication.
3. Check if authentication data is transmitted over insecure channels.
4. If authentication lacks security measures, it indicates vulnerability to insecure authentication.
5. Implement multi-factor authentication and use secure protocols like HTTPS for authentication.

Cookie Theft

Parameters:

- Transmission of cookies over unencrypted channels.
- Lack of secure cookie attributes such as HttpOnly and Secure.
- Vulnerable session management practices.

Steps to Follow:

1. Monitor network traffic for cookie transmission.
2. Check if cookies lack secure attributes like HttpOnly and Secure.
3. Test for vulnerabilities in session management practices.
4. If cookies are easily stolen or manipulated, it indicates vulnerability to cookie theft.
5. Implement secure cookie handling practices and use encryption for transmitting cookies.

Credential Reuse

Parameters:

- Lack of password rotation policies.
- Use of the same credentials across multiple accounts.
- Absence of monitoring for credential reuse attacks.

Steps to Follow:

1. Review password management policies and practices.
2. Check for the use of identical credentials across multiple accounts.
3. Test if passwords are reused without rotation.

4. Monitor for patterns of credential reuse across accounts.
5. If credential reuse is observed, it indicates vulnerability to credential reuse attacks.
6. Implement password rotation policies and monitor for credential reuse to mitigate the risk.

Sensitive Data Exposure

Inadequate Encryption

Parameters:

- Lack of encryption for sensitive data transmission.
- Use of weak encryption algorithms or outdated protocols.
- Absence of encryption for data at rest.
- Failure to implement encryption key management practices.

Steps to Follow:

1. Review data transmission protocols and check if sensitive data is transmitted in plain text.
2. Assess encryption algorithms and protocols used for their strength and security.
3. Check if data at rest is stored without encryption.
4. Examine encryption key management practices to ensure proper handling and protection of encryption keys.
5. If any of these conditions are met, it indicates vulnerability to inadequate encryption.

Insecure Direct Object References (IDOR)

Parameters:

- Direct access to objects or resources without proper authorization.
- Lack of access controls or authorization checks.
- Exposure of sensitive data through predictable object references.

Steps to Follow:

1. Identify resources or objects accessible through direct references.
2. Test if authorization checks are enforced for accessing these resources.
3. Attempt to access sensitive data by manipulating object references or URLs.
4. If unauthorized access to sensitive data is possible through direct object references, it indicates vulnerability to IDOR.
5. Implement proper access controls and authorization checks to prevent IDOR attacks.

Data Leakage

Parameters:

- Improper handling of sensitive data.
- Lack of data masking or anonymization techniques.
- Vulnerable data storage or transmission practices.
- Insufficient access controls on data.

Steps to Follow:

1. Review data handling processes and check for any mishandling of sensitive data.
2. Assess if sensitive data is adequately masked or anonymized to protect privacy.

3. Evaluate data storage and transmission practices for vulnerabilities.
4. Check if access controls are effectively implemented to restrict unauthorized access to data.
5. If sensitive data is leaked or exposed, it indicates vulnerability to data leakage.
6. Implement data protection measures such as encryption, data masking, and access controls to prevent data leakage.

Unencrypted Data Storage

Parameters:

- Storage of sensitive data in plain text or unencrypted format.
- Absence of encryption mechanisms for data storage.
- Failure to secure databases or file systems containing sensitive information.

Steps to Follow:

1. Review data storage practices and check if sensitive data is stored in plain text.
2. Assess if encryption mechanisms are in place to protect data stored in databases or file systems.
3. Evaluate the security measures implemented to safeguard databases or file systems containing sensitive information.
4. If data is stored without encryption, it indicates vulnerability to unencrypted data storage.
5. Implement encryption for data storage and enhance security measures to protect sensitive data from unauthorized access.

Missing Security Headers

Parameters:

- Absence of security-related HTTP headers such as Content Security Policy (CSP), Strict-Transport-Security (HSTS), or X-Content-Type-Options.
- Failure to implement Cross-Origin Resource Sharing (CORS) policies.
- Lack of protection against common web vulnerabilities.

Steps to Follow:

1. Analyze HTTP responses and check for the presence of security-related headers such as CSP, HSTS, and X-Content-Type-Options.
2. Evaluate CORS policies to ensure proper configuration and enforcement.
3. Check if web applications are adequately protected against common vulnerabilities such as XSS, CSRF, and clickjacking.
4. If security headers are missing or misconfigured, it indicates vulnerability to missing security headers.
5. Implement proper security headers and policies to enhance the security posture of web applications.

Insecure File Handling

Parameters:

- Lack of validation for file uploads.
- Execution of files from untrusted sources.
- Vulnerable file access controls.
- Failure to sanitize file names or paths.

Steps to Follow:

1. Review file upload functionality and check if uploaded files are validated for content and file type.
2. Assess if files from untrusted sources are executed without proper validation.
3. Evaluate file access controls to ensure that only authorized users can access or manipulate files.
4. Check if file names or paths are sanitized to prevent directory traversal attacks.
5. If file handling practices are insecure, it indicates vulnerability to insecure file handling.
6. Implement secure file upload processes, validate file contents, and enforce proper access controls to mitigate the risk of exploitation.

Security Misconfiguration

Default Passwords

Parameters:

- Use of default or easily guessable passwords for accounts or systems.
- Failure to change default passwords after installation or deployment.
- Lack of password complexity requirements.
- Absence of password expiration policies.

Steps to Follow:

1. Review system or device documentation for default passwords.
2. Check if default passwords are still in use after installation or deployment.
3. Assess password policies for complexity requirements and expiration settings.
4. Test for weak or easily guessable passwords across accounts or systems.
5. If default or weak passwords are found, it indicates vulnerability to default passwords.
6. Implement strong password policies, enforce password changes, and avoid using default passwords to mitigate the risk.

Directory Listing

Parameters:

- Publicly accessible directories that list the contents of a directory.
- Absence of index files or default pages in web server configurations.
- Lack of access controls on directory listings.
- Disclosure of sensitive information through directory listings.

Steps to Follow:

1. Navigate to directories on web servers and check if directory listings are displayed.
2. Review web server configurations for default pages or index files.
3. Test directory access controls to see if listings are restricted.
4. Look for sensitive information exposed through directory listings.
5. If directories are publicly accessible and list contents, it indicates vulnerability to directory listing.
6. Implement access controls and disable directory listings to prevent disclosure of sensitive information.

Unprotected API Endpoints

Parameters:

- Exposure of APIs without authentication or authorization mechanisms.
- Lack of encryption for data transmitted via APIs.
- Absence of rate limiting or throttling on API requests.
- Disclosure of sensitive information through APIs.

Steps to Follow:

1. Identify APIs exposed to the public or accessible without authentication.
2. Test if APIs require authentication or authorization for access.
3. Check if data transmitted via APIs is encrypted.
4. Assess if rate limiting or throttling mechanisms are in place for API requests.
5. Look for sensitive information exposed through API responses.
6. If APIs are unprotected and expose sensitive data, it indicates vulnerability to unprotected API endpoints.
7. Implement authentication, encryption, rate limiting, and data protection measures to secure API endpoints.

Open Ports and Services

Parameters:

- Identification of open ports and services on network devices.
- Unnecessary or unauthorized services running on systems.
- Lack of port filtering or firewall rules to restrict access.
- Exposure of vulnerable services to potential attacks.

Steps to Follow:

1. Conduct port scanning on network devices to identify open ports.
2. Review system configurations to identify running services.
3. Test for port filtering or firewall rules to restrict access.
4. Look for vulnerable services running on open ports.
5. If unnecessary or vulnerable services are exposed, it indicates vulnerability to open ports and services.
6. Implement port filtering, firewall rules, and disable unnecessary services to reduce the attack surface.

Improper Access Controls

Parameters:

- Insufficient authentication mechanisms.
- Lack of authorization checks on resources or functionalities.
- Failure to enforce least privilege principles.
- Disclosure of sensitive information due to inadequate access controls.

Steps to Follow:

1. Review authentication mechanisms to ensure they are robust and secure.
2. Assess if proper authorization checks are implemented for accessing resources.
3. Check if least privilege principles are enforced to restrict access.
4. Look for instances where sensitive information is accessible without proper access controls.
5. If access controls are inadequate, it indicates vulnerability to improper access controls.
6. Implement strong authentication, authorization mechanisms, and least privilege principles to enhance access controls and protect sensitive information.

Information Disclosure

Parameters:

- Exposure of sensitive information in error messages or responses.
- Disclosure of file paths, stack traces, or system configurations.
- Lack of proper error handling and masking techniques.
- Insecure transmission of sensitive data.

Steps to Follow:

1. Trigger error conditions in web applications or systems to observe error messages.
2. Review error messages for any disclosure of sensitive information.
3. Check if file paths, stack traces, or system configurations are exposed in responses.
4. Evaluate error handling practices for vulnerabilities.
5. Test for insecure transmission of sensitive data across network channels.
6. If sensitive information is exposed, it indicates vulnerability to information disclosure.
7. Implement proper error handling, data masking, and encryption techniques to prevent disclosure of sensitive information.

Unpatched Software

Parameters:

- Presence of outdated software versions with known vulnerabilities.
- Failure to apply security patches or updates in a timely manner.
- Absence of vulnerability management processes.
- Exposure to exploitation of known vulnerabilities.

Steps to Follow:

1. Identify software versions installed on systems and compare them with the latest available versions.
2. Check if security patches or updates have been applied to software.
3. Review vulnerability management processes to ensure patches are applied promptly.

4. Assess the risk of exploitation of known vulnerabilities in unpatched software.
5. If outdated or unpatched software is found, it indicates vulnerability to unpatched software.
6. Implement regular patch management processes and stay updated with security advisories to mitigate the risk of exploitation.

Misconfigured CORS

Parameters:

- Improper configuration of Cross-Origin Resource Sharing (CORS) policies.
- Lack of restrictions on cross-origin requests.
- Failure to enforce secure CORS policies.
- Exposure to cross-origin attacks such as CSRF or data theft.

Steps to Follow:

1. Review CORS policies configured on web servers or applications.
2. Check if cross-origin requests are allowed without proper restrictions.
3. Assess if secure CORS policies are enforced to prevent cross-origin attacks.
4. Test for vulnerabilities such as CSRF or data theft due to misconfigured CORS policies.
5. If CORS policies are misconfigured, it indicates vulnerability to misconfigured CORS.
6. Implement secure CORS policies and restrict cross-origin requests to mitigate the risk of cross-origin attacks.

HTTP Security Headers Misconfiguration

Parameters:

- Absence or misconfiguration of security-related HTTP headers such as Content Security Policy (CSP), Strict-Transport-Security (HSTS), or X-Frame-Options.
- Failure to implement proper HTTP security headers.
- Exposure to common web vulnerabilities due to misconfigured headers.

Steps to Follow:

1. Analyse HTTP responses and check for the presence and configuration of security related headers such as CSP, HSTS, and X-Frame-Options.
2. Assess if proper security headers are implemented to mitigate common web vulnerabilities.
3. Review if HTTP security headers are correctly configured to prevent attacks such as XSS, clickjacking, or protocol downgrade attacks.
4. If security headers are missing or misconfigured, it indicates vulnerability to HTTP security headers misconfiguration.
5. Implement proper HTTP security headers and configurations to enhance the security posture of the web application.

XML-Related Vulnerabilities

XML External Entity (XXE) Injection

Parameters:

- Lack of input validation and sanitization for XML input.
- Use of external entities in XML documents without proper restrictions.
- Execution of malicious XML code by attackers.
- Vulnerable XML parsing libraries or frameworks.

Steps to Follow:

1. Identify input fields or parameters that accept XML input.
2. Input payloads containing external entities into these fields.
3. Submit the input and observe if the XML parser processes external entities.
4. If external entities are parsed and executed, it indicates vulnerability to XXE injection.
5. Test for different XXE payloads to exploit the vulnerability.
6. Implement input validation and disable external entity processing to mitigate the risk of XXE injection.

XML Entity Expansion (XEE)

Parameters:

- Lack of limits on entity expansion in XML documents.
- Absence of protections against recursive entity expansion.
- Execution of malicious XML code by attackers.
- Vulnerable XML parsing libraries or frameworks.

Steps to Follow:

1. Identify XML documents or input fields that allow entity expansion.
2. Input payloads containing recursive entity expansion into these fields.
3. Submit the input and observe if the XML parser expands entities recursively.
4. If recursive entity expansion occurs, it indicates vulnerability to XEE.
5. Test for different XEE payloads to exploit the vulnerability.
6. Implement limits on entity expansion and disable recursive entity expansion to mitigate the risk of XEE.

XML Bomb

Parameters:

- Use of XML documents containing maliciously crafted elements.
- Creation of large, nested XML structures designed to consume resources.
- Execution of denial-of-service attacks through XML parsing.

Steps to Follow:

1. Analyze XML documents for large, nested structures or elements.
2. Check if XML documents contain entities designed to consume excessive resources.
3. Submit the XML document to an XML parser and observe resource usage.

4. If the XML parser consumes excessive resources or crashes, it indicates vulnerability to XML bomb attacks.
5. Test for different XML bomb payloads to exploit the vulnerability.
6. Implement measures such as entity expansion limits or XML payload size restrictions to mitigate the risk of XML bomb attacks.

Broken Access Control

Inadequate Authorization

Parameters:

- Lack of proper access controls on resources or functionalities.
- Absence of role-based access control mechanisms.
- Insufficient validation of user permissions.
- Exposure of sensitive functionalities to unauthorized users.

Steps to Follow:

1. Review access control mechanisms to ensure they're properly implemented.
2. Assess if roles and permissions are assigned based on user roles.
3. Test for any gaps or vulnerabilities in user permission validation.
4. Identify sensitive functionalities accessible to unauthorized users.
5. If inadequate authorization is found, it indicates vulnerability to unauthorized access.
6. Implement proper access controls and role-based authorization to mitigate the risk.

Privilege Escalation

Parameters:

- Opportunities for users to elevate their privileges beyond their authorized level.
- Lack of proper validation for privilege escalation attempts.
- Vulnerable authentication mechanisms.
- Exposure of privileged functionalities to unauthorized users.

Steps to Follow:

1. Identify functionalities or processes where users can potentially elevate their privileges.
2. Test for the presence of vulnerabilities that allow unauthorized privilege escalation.
3. Review authentication mechanisms for any weaknesses that could facilitate privilege escalation.
4. Identify any privileged functionalities accessible to unauthorized users.
5. If privilege escalation is possible, it indicates vulnerability to unauthorized privilege escalation.
6. Implement strict validation checks and access controls to prevent unauthorized privilege escalation.

Insecure Direct Object References

Parameters:

- Direct access to objects or resources without proper authorization.
- Lack of access controls or authorization checks.
- Exposure of sensitive data through predictable object references.

Steps to Follow:

1. Identify resources or objects accessible through direct references.
2. Test if authorization checks are enforced for accessing these resources.
3. Check if sensitive data is exposed through predictable object references.
4. If unauthorized access to sensitive data is possible, it indicates vulnerability to insecure direct object references.
5. Implement proper access controls and authorization checks to prevent unauthorized access to sensitive resources.

Forceful Browsing

Parameters:

- Availability of sensitive pages or functionalities without proper access controls.
- Exposure of internal or hidden functionalities to unauthorized users.
- Absence of protection against brute force attempts to access restricted resources.

Steps to Follow:

1. Attempt to access pages or functionalities directly without proper authentication or authorization.
2. Identify any sensitive pages or functionalities accessible through direct browsing.
3. Test for vulnerabilities that allow brute force attempts to access restricted resources.
4. If unauthorized access is possible, it indicates vulnerability to forceful browsing.
5. Implement proper access controls and protection mechanisms to restrict unauthorized access to sensitive resources.

Missing Function-Level Access Control

Parameters:

- Lack of granularity in access controls, leading to overprivileged access.
- Absence of access controls at the function or operation level.
- Exposure of sensitive operations to unauthorized users.

Steps to Follow:

1. Review access control mechanisms to assess the granularity of access permissions.
2. Identify if access controls are applied at the function or operation level.
3. Test for any gaps or vulnerabilities in function-level access controls.
4. Identify sensitive operations accessible to unauthorized users.
5. If access controls lack granularity, it indicates vulnerability to missing function-level access control.
6. Implement access controls at a granular level and restrict access to sensitive operations based on user permissions.

Insecure Deserialization

Remote Code Execution via Deserialization

Parameters:

- Use of insecure deserialization methods or libraries.
- Acceptance of serialized data from untrusted sources.
- Lack of input validation and sanitization for deserialized data.
- Execution of malicious code embedded in serialized objects.

Steps to Follow:

1. Identify deserialization points in the application where serialized data is accepted.
2. Test if serialized data from untrusted sources can be deserialized.
3. Assess if input validation and sanitization are performed on deserialized data.
4. Inject malicious code into serialized objects and observe if it's executed upon deserialization.
5. If code execution occurs, it indicates vulnerability to remote code execution via deserialization.
6. Implement secure deserialization practices and input validation to mitigate the risk.

Data Tampering

Parameters:

- Lack of integrity checks on incoming data.
- Absence of encryption for transmitted data.
- Vulnerable data validation mechanisms.
- Unauthorized modification of data by attackers.

Steps to Follow:

1. Monitor incoming data for any lack of integrity checks or encryption.
2. Test if data validation mechanisms can be bypassed or manipulated.
3. Attempt to modify data in transit or stored data without proper authorization.
4. Assess if unauthorized modifications to data are possible.
5. If data can be tampered with, it indicates vulnerability to data tampering.
6. Implement integrity checks, encryption, and robust validation to prevent data tampering.

Object Injection

Parameters:

- Acceptance of serialized objects from untrusted sources.
- Lack of input validation and sanitization for deserialized objects.
- Execution of malicious code embedded in serialized objects.

Steps to Follow:

1. Identify points in the application where serialized objects are accepted from untrusted sources.
2. Test if input validation and sanitization are performed on deserialized objects.
3. Inject malicious code into serialized objects and observe if it's executed upon deserialization.
4. Check for vulnerabilities allowing unauthorized object injection.

5. If code execution occurs, it indicates vulnerability to object injection.
6. Implement secure deserialization practices and input validation to mitigate the risk.

API Security Issues

Insecure API Endpoints

Parameters:

- Absence of authentication or authorization mechanisms on API endpoints.
- Lack of encryption for data transmitted via APIs.
- Exposure of sensitive functionalities through unprotected API endpoints.

Steps to Follow:

1. Identify API endpoints that lack authentication or authorization mechanisms.
2. Check if data transmitted via APIs is encrypted.
3. Look for sensitive functionalities accessible through unprotected API endpoints.
4. Test API endpoints for unauthorized access or data exposure.
5. If endpoints are insecure, it indicates vulnerability to insecure API endpoints.
6. Implement proper authentication, authorization, and encryption mechanisms to secure API endpoints.

API Key Exposure

Parameters:

- Storage of API keys in client-side code or configuration files.
- Leakage of API keys through network traffic or error messages.
- Lack of protection against API key enumeration attacks.

Steps to Follow:

1. Review client-side code and configuration files for hardcoded API keys.
2. Monitor network traffic for transmission of API keys.
3. Check if error messages or responses contain sensitive API key information.
4. Test for vulnerabilities allowing enumeration of API keys.
5. If API keys are exposed, it indicates vulnerability to API key exposure.
6. Implement secure storage practices, protect against key leakage, and prevent key enumeration to mitigate the risk.

Lack of Rate Limiting

Parameters:

- Absence of rate limiting or throttling mechanisms on API endpoints.
- Exposure to abuse or denial-of-service attacks through excessive API requests.

Steps to Follow:

1. Identify API endpoints that lack rate limiting or throttling mechanisms.
2. Test API endpoints for excessive requests or potential abuse.
3. Check if there are vulnerabilities allowing bypassing of rate limiting controls.
4. Monitor system performance and network traffic for signs of denial-of-service attacks.
5. If rate limiting is not implemented, it indicates vulnerability to lack of rate limiting.

6. Implement rate limiting controls to restrict the number of API requests and prevent abuse.

Inadequate Input Validation

Parameters:

- Lack of validation for input parameters passed to API endpoints.
- Acceptance of malformed or unexpected input data.
- Vulnerable to injection attacks such as SQL injection or XSS due to inadequate validation.

Steps to Follow:

1. Review input parameters accepted by API endpoints.
2. Test for the presence of input validation mechanisms.
3. Provide malformed or unexpected input data to API endpoints and observe system behavior.
4. Check if vulnerabilities such as SQL injection or XSS can be exploited due to inadequate input validation.
5. If input is not properly validated, it indicates vulnerability to inadequate input validation.
6. Implement robust input validation routines to sanitize and validate input data effectively.

Insecure Communication

Man-in-the-Middle (MITM) Attack:

Parameters:

- Abnormal network traffic patterns
- Unusual SSL/TLS certificate changes
- Suspicious redirections or modifications in HTTP requests/responses

Steps to Follow:

1. Monitor network traffic for unusual patterns or unauthorized devices intercepting communication between a client and server.
2. Look for unexpected changes in SSL/TLS certificates, such as self-signed certificates or certificates issued by unknown authorities.
3. Analyze HTTP requests and responses for any signs of redirection or modification, such as unexpected HTTP status codes or content alterations.

Insufficient Transport Layer Security:

Parameters:

- Weak encryption algorithms or cipher suites
- Lack of Perfect Forward Secrecy (PFS)
- Missing or weak certificate validation

Steps to Follow:

1. Scan for weak encryption algorithms or deprecated ciphersuites used in SSL/TLS configurations.
2. Check if the SSL/TLS configuration supports Perfect Forward Secrecy (PFS), which ensures that session keys are not compromised even if long-term private keys are.

3. Verify that SSL/TLS clients validate server certificates properly and reject connections to servers with invalid, expired, or self-signed certificates.

Insecure SSL/TLS Configuration:

Parameters:

- Weak SSL/TLS protocol versions
- Lack of HTTP Strict Transport Security (HSTS)
- Absence of secure cipher suites

Steps to Follow:

1. Identify if SSLv2 or SSLv3 protocols are enabled, as they are known to be insecure and vulnerable to attacks.
2. Check if the web server sends the HTTP Strict Transport Security (HSTS) header to enforce secure connections over HTTPS and prevent downgrade attacks.
3. Ensure that only strong and secure cipher suites are enabled in the SSL/TLS configuration, such as AES-GCM, AES-CBC, or ChaCha20-Poly1305.

Insecure Communication Protocols:

Parameters:

- Use of insecure communication protocols such as HTTP instead of HTTPS
- Lack of encryption for data in transit
- Absence of authentication mechanisms for communication

Steps to Follow:

1. Identify instances where sensitive data is transmitted over insecure protocols such as HTTP, FTP, or Telnet.
2. Monitor network traffic to detect instances where data is transmitted without encryption, leaving it vulnerable to interception and eavesdropping.
3. Ensure that communication protocols implement robust authentication mechanisms to verify the identities of communicating parties and prevent unauthorized access.

Client-Side Vulnerabilities

DOM-based XSS

Parameters:

- Lack of proper input validation and sanitization in client-side scripts.
- Execution of untrusted user input within the Document Object Model (DOM).
- Vulnerability to XSS attacks due to client-side script execution.

Steps to Follow:

1. Review client-side scripts for input handling and manipulation.
2. Identify areas where user input is incorporated into the DOM.
3. Test for vulnerabilities allowing execution of untrusted input within the DOM.
4. Inject XSS payloads into input fields or parameters and observe if they are executed.
5. If untrusted input can execute scripts within the DOM, it indicates vulnerability to DOM-based XSS.

6. Implement strict input validation and output encoding to mitigate XSS risks.

Insecure Cross-Origin Communication

Parameters:

- Lack of proper Cross-Origin Resource Sharing (CORS) policies.
- Absence of Cross-Origin Communication security mechanisms.
- Vulnerability to cross-origin attacks due to insecure communication channels.

Steps to Follow:

1. Review Cross-Origin Resource Sharing (CORS) policies implemented on web pages or APIs.
2. Test for vulnerabilities allowing unauthorized cross-origin communication.
3. Check if Cross-Origin Communication security mechanisms such as Content Security Policy (CSP) are properly configured.
4. Attempt to access resources across different origins and observe if access is allowed.
5. If unauthorized cross-origin communication is possible, it indicates vulnerability to insecure cross-origin communication.
6. Implement strict CORS policies and utilize security mechanisms like CSP to restrict cross-origin communication and prevent attacks.

Browser Cache Poisoning

Parameters:

- Lack of proper cache control headers or directives.
- Vulnerability to cache poisoning attacks due to caching of malicious content.
- Absence of protections against cache poisoning vulnerabilities.

Steps to Follow:

1. Analyze cache control headers or directives sent by web servers in HTTP responses.
2. Test if caching mechanisms are susceptible to cache poisoning attacks.
3. Check if cached content includes potentially malicious or unauthorized data.
4. Attempt to poison the browser cache by injecting malicious content into cached resources.
5. If the browser cache can be poisoned with unauthorized data, it indicates vulnerability to browser cache poisoning.
6. Implement proper cache control headers, validate cached content, and apply protections against cache poisoning attacks to mitigate the risk.

Clickjacking

Parameters:

- Lack of protection against clickjacking attacks such as frame busting.
- Absence of X-Frame-Options header or frame-ancestors directive.
- Vulnerability to UI redressing attacks through clickjacking.

Steps to Follow:

1. Analyze web pages to determine if they are vulnerable to clickjacking attacks.
2. Test for the presence of frame busting techniques or protection mechanisms.
3. Check if X-Frame-Options header or frame-ancestors directive is properly implemented.

4. Attempt to overlay deceptive content on top of legitimate web pages and trick users into clicking on hidden elements.
5. If web pages can be manipulated to perform unintended actions, it indicates vulnerability to clickjacking.
6. Implement frame busting techniques, set X-Frame-Options header, or use frame-ancestors directive to prevent clickjacking attacks and protect user interactions.

HTML5 Security Issues

Parameters:

- Use of deprecated or insecure HTML5 features.
- Vulnerability to various attacks such as XSS, CSRF, or DOM manipulation.
- Lack of proper security controls for HTML5 elements and APIs.

Steps to Follow:

1. Review HTML5 features and functionalities used in web applications.
2. Test for vulnerabilities associated with HTML5 features, such as XSS or CSRF.
3. Check if proper security controls are implemented for HTML5 elements and APIs.
4. Assess if deprecated or insecure HTML5 features are utilized.
5. If web applications are vulnerable to attacks or lack proper security controls, it indicates HTML5 security issues.
6. Update to secure HTML5 features, implement security controls, and perform thorough security testing to mitigate HTML5-related risks.

Denial of Service (DoS)

Distributed Denial of Service (DDoS)

Parameters:

- Unusual spikes in network traffic or requests to web servers.
- Suspected botnet activity or multiple sources of traffic targeting a single server.
- Overwhelming server resources leading to service disruptions or downtime.

Steps to Follow:

1. Monitor network traffic patterns and look for sudden increases in traffic volume.
2. Analyze server logs for indications of multiple requests from different IP addresses.
3. Assess server performance during peak traffic periods to identify resource exhaustion.
4. Investigate any reports of service disruptions or downtime.
5. If there are signs of coordinated attacks or overwhelming traffic, it indicates vulnerability to DDoS attacks.
6. Implement DDoS mitigation strategies such as traffic filtering, rate limiting, or deploying DDoS protection services to mitigate the risk.

Application Layer DoS

Parameters:

- Targeted attacks on specific application functionalities or endpoints.
- Intentional exploitation of vulnerabilities to exhaust application resources.
- Unavailability of critical application functions due to overload.

Steps to Follow:

1. Identify critical application functionalities or endpoints vulnerable to DoS attacks.
2. Test for vulnerabilities that allow exploitation to exhaust application resources.
3. Monitor application performance during peak usage periods for signs of overload.
4. Investigate any reports of users unable to access critical functions due to application overload.
5. If there are indications of targeted attacks or resource exhaustion, it indicates vulnerability to application layer DoS attacks.
6. Implement measures such as rate limiting, request validation, and load balancing to defend against application layer DoS attacks.

Resource Exhaustion

Parameters:

- Continuous consumption of system resources such as CPU, memory, or disk space.
- Suspected malicious activities or misuse leading to resource depletion.
- System instability or performance degradation due to resource exhaustion.

Steps to Follow:

1. Monitor system resource usage metrics such as CPU utilization, memory consumption, and disk space availability.
2. Analyze system logs for any unusual or sustained resource consumption patterns.
3. Assess system performance and stability for signs of degradation or instability.
4. Investigate any reports of system slowdowns or failures due to resource exhaustion.
5. If resources are consistently depleted or system performance is affected, it indicates vulnerability to resource exhaustion attacks.
6. Implement resource management strategies, such as setting resource limits, optimizing code efficiency, and implementing monitoring and alerting systems to mitigate the risk of resource exhaustion.

Slowloris Attack

Parameters:

- Deliberate slowdown of server response times by sending partial HTTP requests.
- Attempted exploitation of server limitations on concurrent connections.
- Disruption of server operations due to prolonged request processing.

Steps to Follow:

1. Monitor server logs for incomplete or partial HTTP requests from client IP addresses.
2. Analyze server performance metrics for indications of increased response times or server timeouts.
3. Test server response times under varying loads to identify performance bottlenecks.
4. Investigate any reports of users experiencing slow or unresponsive server behavior.
5. If there are signs of prolonged request processing or server slowdowns, it indicates vulnerability to Slowloris attacks.
6. Implement mitigation techniques such as connection rate limiting, server-side timeouts, or deploying intrusion detection systems to detect and prevent Slowloris attacks.

XML Denial of Service

Parameters:

- Exploitation of XML parser vulnerabilities leading to resource exhaustion.
- Sending crafted XML payloads designed to overwhelm parser capabilities.
- Disruption of XML-based services or applications due to processing overload.

Steps to Follow:

1. Analyze XML parser implementations for known vulnerabilities or weaknesses.
2. Test XML parsing functionality with crafted payloads designed to trigger resource exhaustion.
3. Monitor system performance during XML data processing for signs of overload or slowdowns.
4. Investigate any reports of XML-based services or applications becoming unresponsive or failing under load.
5. If XML parsing operations result in resource exhaustion or service disruptions, it indicates vulnerability to XML DoS attacks.
6. Implement secure XML parsing practices, input validation, and limits on XML payload size to defend against XML DoS attacks.

Other Web Vulnerabilities

Server-Side Request Forgery (SSRF)

Parameters:

- Acceptance of user-supplied URLs for server-side requests.
- Lack of input validation for URL parameters.
- Ability to access internal resources through SSRF attacks.
- Exposed sensitive data or services due to unauthorized server-side requests.

Steps to Follow:

1. Identify endpoints or functionalities that accept URLs for server-side requests.
2. Test if input validation is applied to URL parameters.
3. Attempt to access internal resources or sensitive services through manipulated URLs.
4. Check for unauthorized requests made to internal systems or sensitive data.
5. If unauthorized server-side requests are possible, it indicates vulnerability to SSRF attacks.
6. Implement strict input validation and enforce whitelisting of allowed URLs to prevent SSRF vulnerabilities.

HTTP Parameter Pollution (HPP)

Parameters:

- Multiple occurrences of the same parameter in HTTP requests.
- Lack of proper handling for duplicate or conflicting parameters.
- Potential manipulation of application logic or data due to parameter pollution.

Steps to Follow:

1. Analyze HTTP requests for multiple occurrences of the same parameter.

2. Test if the application correctly handles duplicate or conflicting parameters.
3. Attempt to manipulate application behavior or data by polluting parameters with conflicting values.
4. Check for unexpected outcomes or errors resulting from parameter pollution.
5. If application behavior or data can be manipulated through parameter pollution, it indicates vulnerability to HPP.
6. Implement strict parameter handling and validation to prevent parameter pollution attacks.

Insecure Redirects and Forwards

Parameters:

- Use of unvalidated or user-controlled redirect or forward URLs.
- Lack of proper validation for redirect and forward parameters.
- Susceptibility to phishing attacks or unauthorized access through manipulated redirects.

Steps to Follow:

1. Identify redirect or forward functionalities that accept user-controlled URLs.
2. Test if input validation is applied to redirect and forward parameters.
3. Attempt to manipulate redirect or forward URLs to unauthorized destinations.
4. Check for potential phishing attempts or unauthorized access made possible by manipulated redirects.
5. If users can be redirected to unauthorized locations, it indicates vulnerability to insecure redirects and forwards.
6. Implement strict validation of redirect and forward URLs to mitigate the risk of unauthorized access or phishing attacks.

File Inclusion Vulnerabilities

Parameters:

- Inclusion of files from user-supplied or untrusted sources.
- Lack of input validation and sanitization for file inclusion parameters.
- Potential execution of arbitrary code or disclosure of sensitive information.

Steps to Follow:

1. Identify functionalities that include files from user-supplied or untrusted sources.
2. Test if input validation and sanitization are applied to file inclusion parameters.
3. Attempt to include arbitrary files or manipulate file paths to access sensitive resources.
4. Check for unintended file disclosures or execution of arbitrary code resulting from file inclusion.
5. If arbitrary files can be included or sensitive information is exposed, it indicates vulnerability to file inclusion vulnerabilities.
6. Implement strict input validation and enforce restrictions on file inclusion to prevent attacks.

Security Header Bypass

Parameters:

- Absence of or weak implementation of security headers such as Content Security Policy (CSP) or X-Content-Type-Options.

- Vulnerability to various attacks due to lack of proper header enforcement.
- Exposed to XSS, clickjacking, or MIME sniffing attacks.

Steps to Follow:

1. Review implemented security headers such as CSP or X-Content-Type-Options.
2. Test if security headers are properly enforced and configured.
3. Attempt to bypass security controls or exploit vulnerabilities due to weak header implementation.
4. Check for vulnerabilities allowing XSS, clickjacking, or MIME sniffing attacks.
5. If security headers can be bypassed or vulnerabilities are exploited, it indicates vulnerability to security header bypass.
6. Implement secure configurations for security headers and enforce strict header policies to prevent bypass attacks and mitigate associated risks.

Clickjacking

Parameters:

- Lack of protection against clickjacking attacks such as frame busting.
- Absence of X-Frame-Options header or frame-ancestors directive.
- Vulnerability to UI redressing attacks through clickjacking.

Steps to Follow:

1. Analyze web pages to determine if they are vulnerable to clickjacking attacks.
2. Test for the presence of frame busting techniques or protection mechanisms.
3. Check if X-Frame-Options header or frame-ancestors directive is properly implemented.
4. Attempt to overlay deceptive content on top of legitimate web pages and trick users into clicking on hidden elements.
5. If web pages can be manipulated to perform unintended actions, it indicates vulnerability to clickjacking.
6. Implement frame busting techniques, set X-Frame-Options header, or use frame-ancestors directive to prevent clickjacking attacks and protect user interactions.

Inadequate Session Timeout

Parameters:

- Insufficient session timeout settings leading to prolonged session durations.
- Lack of automatic session termination after periods of inactivity.
- Susceptibility to session hijacking or unauthorized access due to extended session lifetimes.

Steps to Follow:

1. Review session management settings and session timeout configurations.
2. Test if sessions remain active for prolonged periods without timeout.
3. Check if sessions are terminated automatically after periods of inactivity.
4. Attempt to hijack active sessions or access restricted resources due to extended session lifetimes.
5. If sessions persist beyond defined timeout periods, it indicates vulnerability to inadequate session timeout.

6. Implement appropriate session timeout settings and automatic session termination mechanisms to mitigate the risk of session hijacking and unauthorized access.

Insufficient Logging and Monitoring

Parameters:

- Lack of comprehensive logging for application activities and security events.
- Inadequate monitoring for suspicious or anomalous behavior.
- Difficulty in detecting and responding to security incidents due to limited visibility.

Steps to Follow:

1. Review logging mechanisms and configurations to ensure they capture relevant application activities and security events.
2. Assess monitoring systems for their ability to detect suspicious or anomalous behavior.
3. Check for gaps in logging coverage or insufficient granularity in log data.
4. Monitor for irregularities in system behavior or security events that go unnoticed due to inadequate logging and monitoring.
5. If there is insufficient visibility into system activities or security events, it indicates vulnerability to inadequate logging and monitoring.
6. Implement comprehensive logging practices, enhance monitoring capabilities, and establish robust incident response procedures to improve visibility and responsiveness to security incidents.

Business Logic Vulnerabilities:

Parameters:

- Deviation from expected business processes
- Inconsistencies in data validation and processing
- Unauthorized access to sensitive functionalities

Steps to Follow:

1. Analyze application workflows and identify any deviations from expected business logic.
2. Test input validation mechanisms to uncover inconsistencies or bypasses in data processing.
3. Perform security testing to ensure that sensitive functionalities are properly protected against unauthorized access.

API Abuse:

Parameters:

- Excessive or abnormal API requests
- Unauthorized access to restricted API endpoints
- Unexpected changes in API usage pattern

Steps to Follow:

1. Monitor API traffic for anomalies such as a high volume of requests from a single source or unusual request patterns.
2. Audit API endpoints to ensure that proper authentication and authorization mechanisms are in place to restrict access to sensitive data and functionalities.

3. Implement rate limiting and access controls to mitigate the risk of API abuse and enforce usage policies.

Mobile Web Vulnerabilities

Insecure Data Storage on Mobile Devices

Parameters:

- Storage of sensitive data in insecure locations on mobile devices, such as unprotected local storage or insecure databases.
- Lack of encryption or obfuscation for stored data on mobile devices.
- Vulnerability to data theft or unauthorized access due to insecure storage practices.

Steps to Follow:

1. Analyze mobile applications for storage of sensitive data on the device.
2. Assess the security measures implemented for data storage, such as encryption and access controls.
3. Test for vulnerabilities allowing unauthorized access to stored data or the presence of sensitive data in insecure locations.
4. Attempt to extract or manipulate stored data through file system access or other means.
5. If sensitive data is stored insecurely or vulnerable to unauthorized access, it indicates a vulnerability in data storage practices on mobile devices.
6. Implement secure storage mechanisms, including encryption and proper access controls, to protect sensitive data from unauthorized access or theft.

Insecure Data Transmission on Mobile Devices

Parameters:

- Transmission of sensitive data over insecure channels, such as unencrypted HTTP connections or insecure network protocols.
- Absence of TLS/SSL encryption for data transmitted between mobile devices and servers.
- Vulnerability to interception and eavesdropping of transmitted data.

Steps to Follow:

1. Monitor network traffic between mobile devices and servers for the transmission of sensitive data.
2. Assess the security measures implemented for data transmission, such as TLS/SSL encryption.
3. Test for vulnerabilities allowing interception or manipulation of transmitted data.
4. Attempt to intercept or eavesdrop on data transmitted over insecure channels.
5. If sensitive data is transmitted without proper encryption or vulnerable to interception, it indicates a vulnerability in data transmission practices on mobile devices.
6. Implement secure communication protocols and enforce TLS/SSL encryption to protect sensitive data during transmission and prevent interception attacks.

Insecure Mobile API Endpoints

Parameters:

- Lack of authentication or authorization mechanisms on mobile API endpoints.

- Exposed sensitive functionalities through unprotected mobile API endpoints.
- Vulnerability to unauthorized access or misuse of mobile APIs.

Steps to Follow:

1. Identify mobile API endpoints that lack authentication or authorization mechanisms.
2. Test if sensitive functionalities are accessible through unprotected mobile API endpoints.
3. Assess the security measures implemented for access control and data protection on mobile APIs.
4. Attempt to access restricted functionalities or manipulate data through insecure mobile API endpoints.
5. If unauthorized access or misuse of mobile APIs is possible, it indicates a vulnerability in mobile API endpoint security.
6. Implement proper authentication, authorization, and access control mechanisms to secure mobile API endpoints and protect against unauthorized access or misuse.

Mobile App Reverse Engineering

Parameters:

- Availability of source code or sensitive information within mobile apps.
- Lack of obfuscation or anti-tampering measures in mobile apps.
- Vulnerability to reverse engineering attacks, including code extraction and analysis.

Steps to Follow:

1. Analyze mobile apps for the presence of sensitive information or code that can be reverse engineered.
2. Assess the level of obfuscation and anti-tampering measures implemented in mobile apps.
3. Test for vulnerabilities allowing extraction or manipulation of source code and sensitive data from mobile apps.
4. Attempt to reverse engineer mobile apps to uncover proprietary algorithms or security vulnerabilities.
5. If source code or sensitive information can be easily extracted or reverse engineered, it indicates a vulnerability in mobile app security.
6. Implement robust obfuscation techniques, anti-tampering measures, and code protection mechanisms to prevent reverse engineering attacks and safeguard intellectual property and sensitive data in mobile apps.

IoT Web Vulnerabilities

Insecure IoT Device Management

Parameters:

- Lack of secure management interfaces for IoT devices, such as insecure web portals or APIs.
- Absence of proper access controls or authentication mechanisms for device management.
- Vulnerability to unauthorized access or manipulation of IoT devices.

Steps to Follow:

1. Identify management interfaces or protocols used for IoT device management.

2. Test if secure authentication mechanisms are in place for accessing device management interfaces.
3. Assess the implementation of access controls to restrict unauthorized access to device management functionalities.
4. Attempt to gain unauthorized access to IoT devices or manipulate device settings through insecure management interfaces.
5. If unauthorized access or manipulation is possible, it indicates vulnerability in IoT device management.
6. Implement secure management interfaces, enforce strong authentication, and apply access controls to protect IoT devices from unauthorized access and manipulation.

Weak Authentication on IoT Devices

Parameters:

- Use of default or weak credentials for accessing IoT devices.
- Lack of password policies or enforcement of strong authentication measures.
- Vulnerability to credential guessing or brute-force attacks on IoT devices.

Steps to Follow:

1. Identify authentication mechanisms used by IoT devices, such as username/password or token-based authentication.
2. Test if default credentials are used or weak password policies are enforced.
3. Assess the strength of authentication mechanisms and their susceptibility to credential guessing or brute-force attacks.
4. Attempt to gain unauthorized access to IoT devices by exploiting weak authentication measures.
5. If unauthorized access is possible due to weak authentication, it indicates vulnerability in IoT device authentication.
6. Implement strong password policies, enforce password complexity requirements, and deploy multi-factor authentication to enhance the security of IoT device authentication.

IoT Device Vulnerabilities

Parameters:

- Presence of known vulnerabilities in IoT device firmware or software.
- Lack of security patches or updates for addressing vulnerabilities.
- Vulnerability to exploitation by malicious actors for unauthorized access or control.

Steps to Follow:

1. Analyze IoT device firmware or software for known vulnerabilities, such as CVE entries or security advisories.
2. Assess the availability of security patches or updates for addressing identified vulnerabilities.
3. Test IoT devices for susceptibility to exploitation by attempting known attack techniques or exploits.
4. Monitor network traffic and device behavior for signs of unauthorized access or control.
5. If vulnerabilities are present and exploitable, it indicates vulnerability in IoT device security.

6. Apply security patches and updates, implement network segmentation, and employ intrusion detection systems to mitigate the risk of IoT device vulnerabilities and unauthorized access.

Web of Things (WoT) Vulnerabilities

Unauthorized Access to Smart Homes

Parameters:

- Weak or default authentication mechanisms for accessing smart home devices.
- Lack of proper access controls or encryption for smart home communication.
- Vulnerability to unauthorized access by attackers or malicious actors.

Steps to Follow:

1. Identify authentication mechanisms used by smart home devices, such as passwords or biometric authentication.
2. Test if default or weak credentials are used, or if authentication is bypassed altogether.
3. Assess the implementation of access controls and encryption for securing smart home communication.
4. Attempt to gain unauthorized access to smart home devices through brute-force attacks, authentication bypasses, or exploiting communication vulnerabilities.
5. If unauthorized access is possible, it indicates vulnerability in smart home security.
6. Implement strong authentication mechanisms, enforce access controls, and use encryption to protect smart home devices from unauthorized access.

IoT Data Privacy Issues

Parameters:

- Inadequate data encryption or anonymization for IoT data transmission and storage.
- Lack of user consent or control over data collection and sharing.
- Vulnerability to data breaches or unauthorized data access.

Steps to Follow:

1. Analyze how IoT data is transmitted and stored and assess the implementation of data encryption and anonymization.
2. Review user consent mechanisms and controls for data collection and sharing.
3. Test for vulnerabilities allowing unauthorized access to IoT data or data breaches.
4. Monitor network traffic and system logs for signs of data leaks or unauthorized access to sensitive data.
5. If data privacy measures are inadequate, it indicates vulnerability in IoT data privacy.
6. Implement strong data encryption, anonymization techniques, and user consent controls to protect IoT data privacy and prevent unauthorized access or data breaches.

Authentication Bypass

Insecure "Remember Me" Functionality

Parameters:

- Weak or inadequate implementation of the "Remember Me" feature in authentication mechanisms.
- Lack of proper session management or expiration for "Remember Me" tokens.
- Vulnerability to unauthorized access or session hijacking.

Steps to Follow:

1. Identify authentication mechanisms that offer a "Remember Me" option for users.
2. Test if proper security measures are implemented for managing "Remember Me" tokens, such as token expiration or secure storage.
3. Assess the susceptibility of "Remember Me" functionality to session hijacking or unauthorized access.
4. Attempt to exploit weaknesses in the "Remember Me" feature to gain unauthorized access or hijack user sessions.
5. If unauthorized access or session hijacking is possible, it indicates vulnerability in the "Remember Me" functionality.
6. Implement robust session management practices, enforce token expiration, and use secure storage mechanisms to mitigate the risk of unauthorized access through the "Remember Me" feature.

CAPTCHA Bypass

Parameters:

- Vulnerabilities in CAPTCHA implementations, such as predictable or weak challenge-response mechanisms.
- Lack of proper validation or enforcement for CAPTCHA responses.
- Vulnerability to automated or manual bypass techniques.

Steps to Follow:

1. Analyze CAPTCHA implementations for weaknesses or vulnerabilities in challenge-response mechanisms.
2. Test if CAPTCHA responses are properly validated and enforced to prevent automated or manual bypass.
3. Assess the susceptibility of CAPTCHA systems to bypass techniques, such as OCR-based attacks or manual exploitation.
4. Attempt to bypass CAPTCHA protections using automated tools or manual methods.
5. If CAPTCHA protections can be bypassed, it indicates vulnerability in CAPTCHA security.
6. Implement stronger challenge-response mechanisms, improve response validation, and use additional security measures to prevent CAPTCHA bypass attacks and enhance overall security.

Server-Side Request Forgery (SSRF)

Blind SSRF

Parameters:

- Lack of immediate feedback or response from the server when exploiting SSRF vulnerabilities.
- Difficulty in detecting SSRF attacks due to blind exploitation techniques.

Steps to Follow:

1. Identify potential SSRF vulnerabilities in the application or network.
2. Test SSRF endpoints or functionalities for lack of immediate responses or feedback.
3. Attempt to access internal resources or services through SSRF attacks.
4. Monitor network traffic and server logs for signs of SSRF exploitation.
5. If SSRF attacks are successful without immediate feedback, it indicates vulnerability to blind SSRF.
6. Implement security controls and monitoring mechanisms to detect and prevent blind SSRF attacks, such as network firewalls, input validation, and comprehensive logging.

Time-Based Blind SSRF

Parameters:

- Lack of immediate feedback or response from the server when exploiting SSRF vulnerabilities.
- Difficulty in detecting SSRF attacks due to time-based exploitation techniques.

Steps to Follow:

1. Identify potential SSRF vulnerabilities in the application or network.
2. Test SSRF endpoints or functionalities for lack of immediate responses or feedback.
3. Attempt time-based exploitation techniques to detect SSRF vulnerabilities.
4. Monitor server response times or delays during SSRF exploitation attempts.
5. If SSRF attacks result in delayed responses or timeouts, it indicates vulnerability to time-based blind SSRF.
6. Implement security controls and monitoring mechanisms to detect and prevent time-based blind SSRF attacks, such as rate limiting, timeout thresholds, and anomaly detection.

Content Spoofing

MIME Sniffing

Parameters:

- Lack of proper MIME type validation for user-supplied files.
- Absence of Content-Disposition headers or strict file type enforcement.
- Vulnerability to content type misinterpretation or execution of malicious files.

Steps to Follow:

1. Analyze file upload functionalities or content-serving endpoints for MIME type validation.
2. Test if files are served without proper Content-Disposition headers or MIME type validation.

3. Attempt to upload or serve files with incorrect or unexpected MIME types.
4. Monitor server responses or file processing for signs of MIME sniffing behavior.
5. If files are processed or executed based on inferred MIME types, it indicates vulnerability to MIME sniffing.
6. Implement strict MIME type validation, enforce Content-Disposition headers, and restrict file execution to prevent MIME sniffing attacks and protect against malicious file execution.

X-Content-Type-Options Bypass

Parameters:

- Lack of proper implementation or enforcement of X-Content-Type-Options header.
- Absence of server-side validation for content types.
- Vulnerability to MIME sniffing or content type misinterpretation attacks.

Steps to Follow:

1. Review server configurations and HTTP responses for the presence of X-Content-Type-Options header.
2. Test if the header is properly enforced and prevents MIME sniffing.
3. Attempt to bypass content type restrictions by manipulating file extensions or content.
4. Monitor browser behavior or server responses for signs of MIME type inference or content type bypass.
5. If content types are inferred or bypassed, it indicates vulnerability to X-Content-Type-Options bypass.
6. Implement strict server-side validation, enforce X-Content-Type-Options header with the "nosniff" directive, and avoid relying on MIME type inference to prevent content type bypass attacks and enhance security.

Content Security Policy (CSP) Bypass

Parameters:

- Inadequate implementation or misconfiguration of Content Security Policy directives.
- Lack of proper enforcement or coverage for protecting against XSS attacks.
- Vulnerability to script execution or injection through policy bypasses.

Steps to Follow:

1. Review web application configurations and HTTP headers for the presence of Content Security Policy (CSP) directives.
2. Test if CSP directives are properly configured and enforced to prevent XSS attacks.
3. Attempt to bypass CSP restrictions by exploiting misconfigurations or policy weaknesses.
4. Monitor browser behavior or server responses for signs of CSP bypass or policy violations.
5. If scripts are executed or injected despite CSP directives, it indicates vulnerability to CSP bypass.
6. Implement strict CSP configurations, enforce security headers, and regularly audit and update CSP policies to mitigate the risk of XSS attacks and prevent CSP bypasses.

Business Logic Flaws

Inconsistent Validation

Parameters:

- Lack of consistent validation checks across different parts of the application.
- Variation in validation rules or behavior leading to inconsistent data handling.
- Vulnerability to data integrity issues or unexpected behavior.

Steps to Follow:

1. Review validation mechanisms and rules implemented throughout the application.
2. Test various input scenarios across different functionalities or modules.
3. Identify inconsistencies or discrepancies in validation behavior or outcomes.
4. Attempt to exploit variations in validation rules to manipulate or bypass input checks.
5. Monitor for unexpected behavior or data integrity issues resulting from inconsistent validation.
6. If validation rules are inconsistent or varied, it indicates vulnerability to inconsistent validation.
7. Implement standardized validation processes, ensure consistency in validation rules, and conduct thorough testing to maintain data integrity and prevent vulnerabilities arising from inconsistent validation.

Race Conditions

Parameters:

- Concurrent access to shared resources or critical sections of code.
- Lack of proper synchronization or locking mechanisms in multi-threaded or distributed systems.
- Vulnerability to data corruption or unauthorized access due to race conditions.

Steps to Follow:

1. Identify areas of the application where multiple processes or threads access shared resources concurrently.
2. Review synchronization mechanisms and locking strategies implemented in multi-threaded or distributed systems.
3. Test scenarios where multiple processes or threads compete for access to critical resources.
4. Attempt to manipulate timing or race conditions to gain unauthorized access or corrupt data.
5. Monitor for unexpected behavior or data inconsistencies resulting from race conditions.
6. If data corruption or unauthorized access occurs under concurrent access, it indicates vulnerability to race conditions.
7. Implement proper synchronization mechanisms, use transactional processing, and enforce access controls to mitigate the risk of race conditions and maintain data integrity.

Order Processing Vulnerabilities

Parameters:

- Vulnerabilities in the order processing workflow, such as insufficient validation of order data.

- Lack of authorization or access controls for order processing functionalities.
- Vulnerability to order manipulation or unauthorized transactions.

Steps to Follow:

1. Analyze the order processing workflow and associated functionalities in the application.
2. Test if order data is validated properly for authenticity, integrity, and authorization.
3. Review access controls and permissions for order processing functionalities.
4. Attempt to manipulate order data or bypass authorization controls to place unauthorized orders.
5. Monitor for unexpected changes in order status or unauthorized transactions.
6. If orders can be manipulated or unauthorized transactions occur, it indicates vulnerability in order processing.
7. Implement robust validation checks, enforce strict authorization controls, and implement transactional safeguards to prevent order processing vulnerabilities and unauthorized transactions.

Price Manipulation

Parameters:

- Vulnerabilities in the pricing mechanism, such as lack of input validation or insufficient authorization checks.
- Lack of audit trails or monitoring for price changes.
- Vulnerability to price manipulation or unauthorized discounts.

Steps to Follow:

1. Review the pricing mechanism and associated functionalities in the application.
2. Test if input data for pricing is properly validated and authorized.
3. Assess the implementation of audit trails or monitoring for price changes.
4. Attempt to manipulate pricing data or bypass authorization controls to change prices.
5. Monitor for unexpected changes in prices or unauthorized discounts.
6. If prices can be manipulated or unauthorized discounts occur, it indicates vulnerability in the pricing mechanism.
7. Implement strict input validation, enforce authorization controls, and implement comprehensive monitoring for price changes to prevent price manipulation vulnerabilities and unauthorized discounts.

Account Enumeration

Parameters:

- Lack of proper error handling or response differentiation for valid and invalid account identifiers.
- Differential responses or timing variations for existing and non-existing accounts.
- Vulnerability to account enumeration or username harvesting.

Steps to Follow:

1. Test account-related functionalities such as login, registration, or password reset for differential responses.

2. Analyze error messages or responses to determine if they provide clues about account validity.
3. Monitor response times or behaviour for timing differences between valid and invalid account identifiers.
4. Attempt to enumerate user accounts or harvest usernames by exploiting differential responses or timing variations.
5. Monitor for abnormal patterns of account access or enumeration attempts.
6. If account enumeration is possible or timing differences exist, it indicates vulnerability to account enumeration.
7. Implement standardized error responses, avoid timing variations, and enforce consistent error handling to prevent account enumeration vulnerabilities and protect user privacy.

User-Based Flaws

Parameters:

- Observing user behaviour and interactions within the system.
- Analysing access controls and permissions.
- Reviewing password policies and data handling procedures.

Steps to Follow:

1. Conduct user training and awareness programs to educate users about security best practices.
2. Implement strong access controls and enforce least privilege principles.
3. Regularly audit and review user activities and permissions.
4. Test user susceptibility to social engineering attacks through simulated scenarios.
5. Monitor for abnormal user behaviors or security incidents stemming from user actions.

Zero-Day Vulnerabilities

Unknown Vulnerabilities

Parameters:

- Conducting proactive security measures such as vulnerability scanning and penetration testing.
- Staying informed about emerging threats and security advisories.
- Employing robust monitoring and anomaly detection systems.

Steps to Follow:

1. Implement vulnerability scanning and penetration testing to identify potential weaknesses before they are exploited.
2. Stay updated with security advisories and threat intelligence sources to understand potential areas of risk.
3. Deploy robust monitoring and anomaly detection systems to detect any suspicious behavior or indicators of compromise.
4. Regularly assess system configurations and architectures for potential vulnerabilities.

Unpatched Vulnerabilities

Parameters:

- Monitoring vendor and security advisories for available patches and updates.
- Assessing system configurations and patch management processes.
- Conducting vulnerability assessments and prioritizing patching based on risk.

Steps to Follow:

1. Establish a patch management process to regularly apply vendor-supplied patches and updates.
2. Monitor vendor and security advisories to stay informed about available patches.
3. Conduct vulnerability assessments to identify unpatched vulnerabilities and prioritize patching based on risk.
4. Implement change management procedures to ensure timely and effective patch deployment.

Day-Zero Exploits

Parameters:

- Monitoring security research and threat intelligence sources for emerging vulnerabilities.
- Assessing system configurations and architecture for potential attack vectors.
- Deploying intrusion detection and prevention systems.

Steps to Follow:

1. Stay updated with security research, threat intelligence sources, and vulnerability databases to identify emerging vulnerabilities.
2. Assess system configurations and architectures to understand potential attack vectors.
3. Deploy intrusion detection and prevention systems to detect and block exploit attempts.
4. Implement security controls and mitigations to reduce the impact of day-zero exploits, such as network segmentation and application firewalls.