Security Testing

Topics

What is security testing?

Security testing is a process of evaluating a system or application to identify and address potential vulnerabilities and security risks. It involves assessing the security posture of the system by simulating attacks, analyzing security controls, and identifying weaknesses that could be exploited by attackers. The primary goal of security testing is to ensure that the system or application is resilient to security threats and meets security requirements.

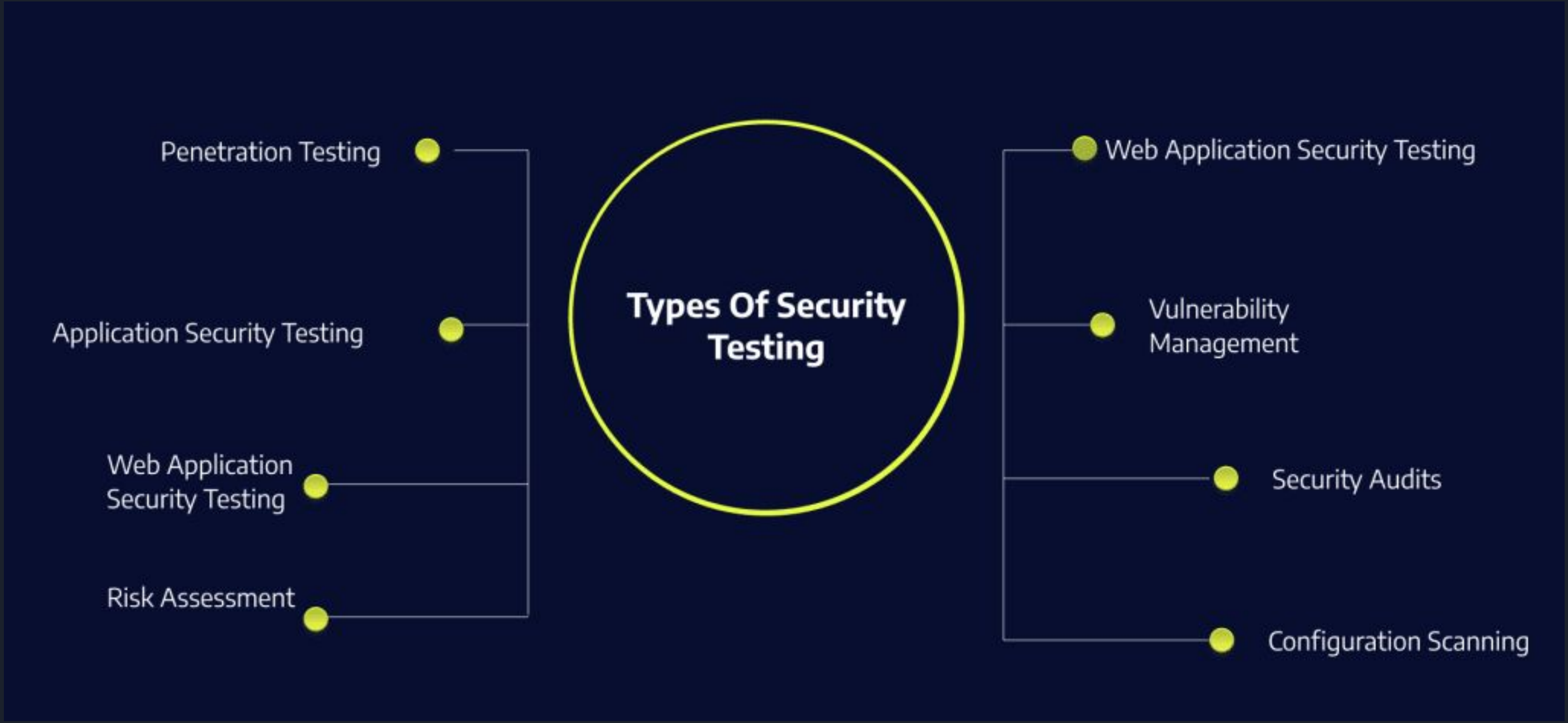
Key principles of security testing

1. Confidentiality
2. Integrity
3. Authentication
4. Authorization
5. Availability

Understanding importance of Security testing

- Risk Reduction
- Customer Satisfaction
- Cost Savings
- Performance Validation
- Bug Detection
- Quality Assurance
- Identifying Vulnerabilities

Types Of Security Testing


Types Of Security Testing
Penetration Testing
Application Security Testing
Web Application Security Testing
Risk Assessment
Web Application Security Testing
Vulnerability Management
Security Audits
Configuration Scanning

Penetration Testing (Ethical Hacking)

Penetration testing is the process of stimulating real-life cyber attacks against an application, software, system, or network under safe conditions. It can help evaluate how existing security measures will measure up in a real attack. Most importantly, penetration testing can find unknown vulnerabilities, including zero-day threats and business logic vulnerabilities.

Application Security Testing (AST)

Application security testing describes methods organizations can use to find and eliminate vulnerabilities in software applications. These methods involve testing, analyzing, and reporting on the security posture of a software application throughout the software development lifecycle (SDLC).

Web Application Security Testing

Web application penetration testing aims to gather information about a web application, discover system vulnerabilities or flaws, investigate the success of exploiting these flaws or vulnerabilities, and evaluate the risk of web application vulnerabilities.

API Security Testing

API security testing helps identify vulnerabilities in application programming interfaces (APIs) and web services, and assist developers in remediating those vulnerabilities. APIs provide access to sensitive data, and attackers can use them as an entry point to internal systems. Testing APIs rigorously and regularly can protect them from unauthorized access and abuse.

Vulnerability Management

Vulnerability management is a continuous process that enables an organization to identify, assess, report, manage, and remediate security vulnerabilities across endpoints, workloads, and networks. Security teams typically use vulnerability scanning tools to detect vulnerabilities and implement manual or automatic processes to fix them.

Configuration Scanning

Security scanning, also known as configuration scanning, is the process of identifying misconfigurations of software, networks and other computing systems. This type of scanning typically checks systems against a list of best practices, specified by research organizations or compliance standards.

Security Audits

A security audit is a structured process for reviewing/auditing an application/software according to a defined standard. Audits usually involve reviews of code or architectures in light of security requirements, analyzing security gaps, and assessing the security posture of hardware configurations, operating systems, and organizational practices. It also evaluates compliance with regulations and compliance standards.

Security Testing Approaches

Black Box Testing- is where the security tester evaluates a system’s security from the outside without knowing the internal processes generating responses

White Box Testing - In white box testing, the tester designs test cases and tests based on the software’s source code.

Gray Box Testing- is a hybrid of white box and black box testing

Security Testing Tools

Static Application Security Testing (SAST)

Dynamic Application Security Testing (DAST)

Static Application Security Testing (SAST)

SAST tools assess the source code while at rest. The purpose of SAST is to identify exploitable flaws and provide a detailed report including findings and recommendations.

You can run SAST to detect issues in source code, to detect issues such as input validation, numerical errors, path traversals, and race conditions. SAST can also be used on compiled code, but this requires binary analyzers.

Dynamic Application Security Testing (DAST)

DAST tools examine the application during runtime. The purpose of DAST is to detect exploitable flaws in the application while it is running, using a wide range of attacks.

You can run DAST checks to check a wide range of components, including scripting, sessions, data injection, authentication, interfaces, responses, and requests

Common JavaScript Security Vulnerabilities

Cross-Site Scripting (XSS)

Cross-Site Request Forgery

Injection attacks

Sensitive Data Exposure

DoS attacks

Client-Side Validation Bypass

Javascript Security Testing Best Practices

Code Review and Static Analysis - Regularly review the codebase for security vulnerabilities. Tools like ESLint, SonarQube, or CodeQL can help identify potential issues statically.

Ensure that sensitive information like API keys, passwords, and tokens are not hard-coded in the source code.

Javascript Security Testing Best Practices

Input Validation and Sanitization - Validate and sanitize all user inputs to prevent injection attacks such as SQL injection, NoSQL injection, and Cross-Site Scripting (XSS).

Use libraries like express-validator or Joi to validate and sanitize input data.

Javascript Security Testing Best Practices

Authentication and Authorization - Implement secure authentication mechanisms like JSON Web Tokens (JWT) for user authentication.

Ensure proper authorization checks are in place to restrict access to resources based on user roles and permissions.

Javascript Security Testing Best Practices

Secure Communication - Use HTTPS protocol to encrypt data transmitted between the client and server.

Implement Cross-Origin Resource Sharing (CORS) policies to restrict unauthorized access to resources.

Javascript Security Testing Best Practices

Database Security - Use parameterized queries or ORM libraries with built-in protection against SQL injection attacks.

Apply the principle of least privilege by granting minimal permissions required for database access.

Javascript Security Testing Best Practices

Database Security - Use parameterized queries or ORM libraries with built-in protection against SQL injection attacks.

Apply the principle of least privilege by granting minimal permissions required for database access.

Javascript Security Testing Best Practices

File Upload Security - Validate file uploads to prevent malicious files from being uploaded.

Restrict file types and sizes based on application requirements.

Store uploaded files outside the web root directory to prevent direct access.

Javascript Security Testing Best Practices

Error Handling - Implement proper error handling to avoid leaking sensitive information to attackers.

Use generic error messages to avoid exposing implementation details to users.

Javascript Security Testing Best Practices

Dependency Management - Regularly update dependencies to patch security vulnerabilities.

Use tools like npm audit or Snyk to scan for known vulnerabilities in npm packages.

Javascript Security Testing Best Practices

Penetration Testing - Conduct regular penetration testing to identify and mitigate security weaknesses.

Utilize tools like OWASP ZAP or Burp Suite for automated security testing.

Thank you !!!