TYPE OF PENTEST

Source Code </>

RedSentry

What is Source Code Analysis?

Source code analysis is the process of **reviewing application code** to identify potential security vulnerabilities. It can be conducted with access to the source code (white-box testing) and often includes both automated and manual techniques to ensure that vulnerabilities related to code quality, logic flaws, and security misconfigurations are identified and mitigated. We currently utilize the **OWASP Secure Coding Practices** as a framework for part of the testing.

1 | Automated Code Scanning phase

In this phase, automated tools are used to quickly identify common and well-documented vulnerabilities in the code.

Static Code Analysis

Run automated scanners to detect issues such as injection flaws, insecure data handling, and hardcoded secrets.

Dependency Scanning

Analyze third-party libraries for known vulnerabilities using tools that identify outdated libraries or publicly known vulnerabilities (e.g., CVEs).

| Examples of Vulnerabilities Found

> **SQL Injection** – Unparameterized SQL queries using user input directly.

> **Insecure Cryptography** – Use of weak cryptographic algorithms, such as MD5 or SHA-1.

> **Sensitive Data Exposure** – Hardcoded credentials or sensitive data within the codebase.

> **Insecure Dependencies** – Outdated libraries with known vulnerabilities.

For the static code analysis, tools like **SonarQube**, **Fortify**, **Checkmarx**, and **Semgrep** can be used.

2 | Manual Code Review

Manual code review is a critical phase where security experts analyze the code for **complex logic flaws**, which automated tools often miss.

Review Key Modules

Focus on high-risk areas such as authentication, authorization, and data validation functions.

Business Logic Analysis

Identify vulnerabilities caused by flawed business logic, such as improper access controls or privilege escalation.

Security-Specific Checks

Validate secure coding practices in input handling, error handling, and session management.

| Examples of Vulnerabilities Found

> **Authorization Bypass** – Inconsistent or missing access controls allowing privilege escalation.

> **Logic Flaws** – Errors in business logic allowing unintended access or manipulation of data.

> **Session Management Issues** – Weak or missing session expiration, improper session token management.

3 | Exploitation phase

In this phase, we use the insights gained from traffic analysis to simulate attacks and demonstrate potential security risks. Key activities include:

Session Hijacking

> Using tools like **Ettercap** or **Burp Suite** to intercept session tokens and impersonate legitimate users.

> Exploiting cleartext or weakly encrypted sessions to gain **unauthorized access**.

Credential Cracking and Reuse

> Extracting captured credentials for **offline cracking** using tools like **Hashcat** or John the **Ripper**.

> Testing captured credentials for **reuse** across other systems in the network.

Man-in-the-Middle (MitM) Attacks

> Setting up a rogue device or service to intercept and manipulate network traffic.

> Exploiting insecure protocols (e.g., ARP spoofing or DNS poisoning) to redirect traffic through malicious systems.

Ingress Route Exploitation

> **Testing alternate routes** for ingress identified in **routing analysis**, such as **VPNs**, **misconfigured firewalls**, or **poorly secured interfaces**.

> Verifying if **traffic segmentation** can be bypassed to access sensitive areas of the network. The goal here is to demonstrate how an attacker could exploit identified vulnerabilities to compromise the network further.

4 | Reporting phase

After completing the penetration test, we compile all findings into a **detailed report**, following Red Sentry's standardized template.

- A summary of the identified vulnerabilities
- The methods used to exploit them
- Recommendations for remediation

This methodology ensures a comprehensive assessment of the source code provided by the client to identify and address any vulnerabilities before they can be exploited by malicious actors.