CODE-EPIC

Code Epic Technologies

More About Penetration Testing

Surviving technology

cr4sh.m4d0v3r
https://code-epic.github.io

June 2022

1 Testing

Penetration Testing (pentesting) is carried out as if the tester was a malicious external attacker with a goal of breaking into the system and either stealing data or carrying out some sort of denial-of-service attack.

Pentesting has the advantage of being more accurate because it has fewer false positives (results that report a vulnerability that isn't actually present), but can be time-consuming to run.

Pentesting is also used to test defence mechanisms, verify response plans, and confirm security policy adherence.

Automated pentesting is an important part of continuous integration validation. It helps to uncover new vulnerabilities as well as regressions for previous vulnerabilities in an environment which quickly changes, and for which the development may be highly collaborative and distributed.

Both manual and automated pentesting are used, often in conjunction, to test everything from servers, to networks, to devices, to endpoints. This document focuses on web application or web site pentesting

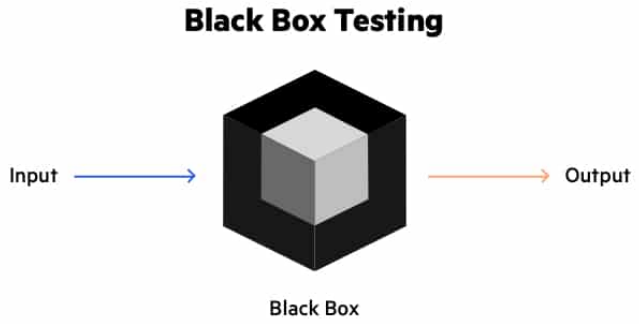
Pentesting usually follows these stages:

1. Explore – The tester attempts to learn about the system being tested. This includes trying to determine what software is in use, what endpoints exist, what patches are installed, etc. It also includes searching the site for hidden content, known vulnerabilities, and other indications of weakness.

2. Attack – The tester attempts to exploit the known or suspected vulnerabilities to prove they exist.

3. Report – The tester reports back the results of their testing, including the vulnerabilities, how they exploited them and how difficult the exploits were, and the severity of the exploitation.

2 Technique



The black box is a powerful technique to check the application under test from the user's perspective. Black box testing is used to test the system against external factors responsible for software failures. This testing approach focuses on the input that goes into the software, and the output that is produced.

Test procedures

Specific knowledge of the application's code, internal structure and programming knowledge in general is not required. The tester is aware of what the software is supposed to do but is not aware of how it does it. For instance, the tester is aware that a particular input returns a certain, invariable output but is not aware of how the software produces the output in the first place.

Test Cases

Test cases are built around specifications and requirements, i.e., what the application is supposed to do. Test cases are generally derived from external descriptions of the software, including specifications, requirements and design parameters. Although the tests used are primarily functional in nature, non-functional tests may also be used. The test designer selects both valid and invalid inputs and determines the correct output, often with the help of a test oracle or a previous result that is known to be good, without any knowledge of the test object's internal structure.

Test design techniques

Typical black-box test design techniques include:

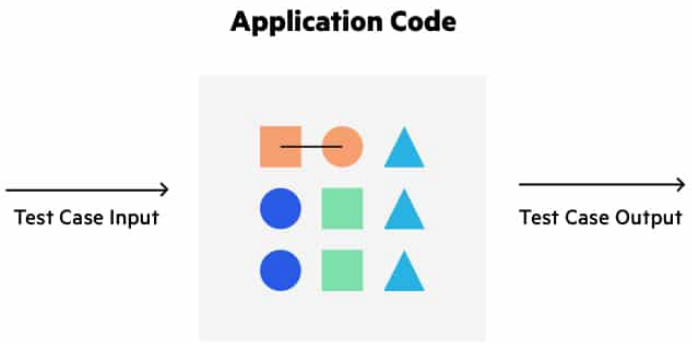
- Decision table testing
- All-pairs testing
- Boundary value analysis
- Cause–effect graph
- Error guessing
- State transition testing
- Use case testing
- User story testing
- Domain analysis
- Syntax testing

Pros	Cons
Testers do not require technical knowledge, programming or IT skills	Difficult to automate
Testers do not need to learn implementation details of the system	Requires prioritization, typically infeasible to test all user paths
Tests can be executed by crowdsourced or outsourced testers	Difficult to calculate test coverage
Low chance of false positives	If a test fails, it can be difficult to understand the root cause of the issue
Tests have lower complexity, since they simply model common user behavior	Tests may be conducted at low scale or on a non-production-like environment

Table 1: Black Box Testing Pros and Cons

Hacking In penetration testing, black-box testing refers to a method where an ethical hacker has no knowledge of the system being attacked. The goal of a black-box penetration test is to simulate an external hacking or cyber warfare attack.

White Box



White box testing is an approach that allows testers to inspect and verify the inner workings of a software system—its code, infrastructure, and integrations with external systems. White box testing is an essential part of automated build processes in a modern Continuous Integration/Continuous Delivery (CI/CD) development pipeline.

Pros	Cons
Ability to achieve complete code coverage	Requires a large effort to automate
Easy to automate	Sensitive to changes in code base, automation requires expensive maintenance
Reduces communication overhead between testers and developers	Cannot test expected functionality that does not exist in the codebase
Allows for continuous improvement of code and development practices	Cannot test from the user's perspective

Table 2: White Box Testing Pros and Cons

White box testing is often referenced in the context of Static Application Security Testing (SAST), an approach that checks source code or binaries automatically and provides feedback on bugs and possible vulnerabilities.

Types of White Box Testing

White box testing can take several forms:

- Unit testing — tests written as part of the application code, which test that each component is working as expected.
- Mutation testing — a type of unit testing that checks the robustness and consistency of the code by defining tests, making small, random changes to the code and seeing if the tests still pass.
- Integration testing — tests specifically designed to check integration points between internal components in a software system, or integrations with external systems.
- White box penetration testing — an ethical hacker acts as a knowledgeable insider, attempting to attack an application based on intimate knowledge of its code and environment.
- Static code analysis — automatically identifying vulnerabilities or coding errors in static code, using predefined patterns or machine learning analysis.

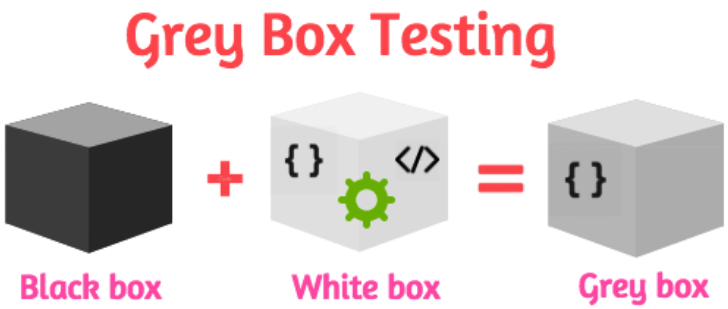
What Does White Box Testing Focus On?

White box tests can focus on discovering any of the following problems with an application's code:

- Security gaps and vulnerabilities — checking to see if security best practices were applied when coding the application, and if the code is vulnerable to known security threats and exploits.
- Broken or poorly structured paths — identifying conditional logic that is redundant, broken or inefficient.
- Expected output — executing all possible inputs to a function to see if it always returns the expected result.

- Loop testing — checking single loops, concatenated loops and nested loops for efficiency, conditional logic, and correct handling of local and global variables.
- Data Flow Testing (DFT) — tracking variables and their values as they pass through the code to find variables that are not correctly initialized, declared but never used, or incorrectly manipulated.

Grey Box Testing



White box testing involves complete knowledge of the inner workings of a system under test and black box involves no knowledge. Grey box testing, however, is a compromise – testing a system with partial knowledge of its internals. It is most commonly used in integration testing, end-to-end system testing, and penetration testing.

Grey box testing combines inputs from developers and testers and can result in more effective testing strategies. It reduces the overhead required to perform functional testing of a large number of user paths, focusing testers on the paths most likely to affect users or result in a defect.

Grey box testing combines the benefits of black box and white box testing

- Ensuring that tests are performed from the user's perspective, like in black box testing.
- Leveraging inside knowledge to focus on the problems that matter most, and to identify and resolve internal weaknesses of the system, like in white box testing.

In the world of Application Security Testing , the grey box testing approach is called Interactive Application Security Testing (IAST). IAST combines:

- SAST — which performs white box testing by evaluating static application code.
- Dynamic Application Security Testing (DAST) — which performs black box testing, by interacting with running applications and discovering faults and vulnerabilities like a user or external attacker would.

Gray Box Testing Techniques

Gray box testing techniques are designed to enable you to perform penetration testing on your applications. These techniques enable you to test for insider threats, such as employees attempting to manipulate applications, and external users, such as attackers attempting to exploit vulnerabilities.

With gray box testing, you can ensure that applications work as expected for authenticated users. You can also verify that malicious users cannot access data or functionality you don't want them to.

When performing gray box testing, there are several techniques you can choose from. Depending on which testing phase you are in and how the application operates, you may want to combine multiple techniques to ensure all potential issues are identified.

Matrix Testing

Matrix testing is a technique that examines all variables in an application. In this technique, technical and business risks are defined by the developers and a list of all application variables are provided. Each variable is then assessed according to the risks it presents. You can use this technique to identify unused or un-optimized variables.

Regression Testing

Regression testing is a technique that enables you to verify whether application changes or bug fixes have caused errors to appear in existing components. You can use it to ensure that modifications to your application

are only improving the product, not relocating faults. When performing regression testing, you need to recreate your tests since inputs, outputs, and dependencies may have changed.

Pattern Testing

Pattern testing is a technique that evaluates past defects to identify patterns that lead to defects. Ideally, these evaluations can highlight which details contributed to defects, how the defects were found, and how effective fixes were. You can then apply this information to identifying and preventing similar defects in new versions of an application or new applications with similar structures.

Orthogonal Array Testing

Orthogonal array testing is a technique you can use when your application has only a few inputs that are too complex or large for extensive testing. This technique enables you to perform test case optimization, where the quality and number of tests performed balance test coverage with effort. This technique is systematic and uses statistics to test pair-based interactions.

3 Application Security Risks

Web Application Security Risks: OWASP Top 10

Software applications can be affected by numerous threats. The Open Web Application Security Project (OWASP) Top 10 list includes critical application threats that are most likely to affect applications in production.

Broken Access Control

Broken access control allows threats and users to gain unauthorized access and privileges. Here are the most common issues:

- It enables attackers to gain unauthorized access to user accounts and act as administrators or regular users.
- It provides users with unauthorized privileged functions.

You can remediate this issue by implementing strong access mechanisms that ensure each role is clearly defined with isolated privileges.

Cryptographic Failures

Cryptographic failures (previously referred to as "sensitive data exposure") occur when data is not properly protected in transit and at rest. It can expose passwords, health records, credit card numbers, and personal data.

This application security risk can lead to non-compliance with data privacy regulations, such as the EU General Data Protection Regulation (GDPR), and financial standards like PCI Data Security Standards (PCI DSS).

Injection (Including XSS, LFI, and SQL Injection)

Injection vulnerabilities enable threat actors to send malicious data to a web application interpreter. It can cause this data to be compiled and executed on the server. SQL injection is a common form of injection.

Insecure Design

Insecure design covers many application weaknesses that occur due to ineffective or missing security controls. Applications that do not have basic security controls capable of against critical threats. While you can fix implementation flaws in applications with secure design, it is not possible to fix insecure design with proper configuration or remediation.

Security Misconfiguration (Including XXE)

Security misconfigurations occur due to a lack of security hardening across the application stack. Here are common security misconfigurations:

- Improperly configuring cloud service permissions
- Leaving unrequired features enabled or installed
- Using default passwords or admin accounts
- XML External Entities (XXE) vulnerabilities
- Learn more in the detailed guide to XML External Entities (XXE)

Vulnerable and Outdated Components

Vulnerable and outdated components (previously referred to as "using components with known vulnerabilities") include any vulnerability resulting from outdated or unsupported software. It can occur when you build or use an application without prior knowledge of its internal components and versions.

Identification and Authentication Failures

Identification and authentication failures (previously referred to as "broken authentication") include any security problem related to user identities. You can protect against identity attacks and exploits by establishing secure session management and setting up authentication and verification for all identities.

Software and Data Integrity Failures

Software and data integrity failures occur when infrastructure and code are vulnerable to integrity violations. It can occur during software updates, sensitive data modification, and any CI/CD pipeline changes that are not validated. Insecure CI/CD pipelines can result in unauthorized access and lead to supply chain attacks.

Security Logging and Monitoring Failures

Security logging and monitoring failures (previously referred to as "insufficient logging and monitoring") occur when application weaknesses cannot properly detect and respond to security risks. Logging and monitoring are critical to the detection of breaches. When these mechanisms do not work, it hinders the application's visibility and compromises alerting and forensics.

Server Side Request Forgery

Server-side request forgery (SSRF) vulnerabilities occur when a web application does not validate a URL inputted by a user before pulling data from a remote resource. It can affect firewall-protected servers and any network access control list (ACL) that does not validate URLs.

4 API Security Risks: OWASP Top 10

APIs enable communication between different pieces of software. Applications with APIs allow external clients to request services from the application. APIs are exposed to various threats and vulnerabilities. The OWASP compiled a list prioritizing the top 10 API security risks.

Broken Object Level Authorization

APIs often expose endpoints handling object identifiers. It creates a wider attack surface Level Access Control issue. Instead, you should check object level authorization in every function that can access a data source through user inputs.

Broken User Authentication

Incorrectly implemented authentication mechanisms can grant unauthorized access to malicious actors. It enables attackers to exploit an implementation flaw or compromise authentication tokens. Once it occurs, attackers can assume a legitimate user identity permanently or temporarily. As a result, the system's ability to identify a client or user is compromised, which threatens the overall API security of the application.

Excessive Data Exposure

Generic implementations often lead to exposure of all object properties without consideration of the individual sensitivity of each object. It occurs when developers rely on clients to perform data filtering before displaying the information to the user.

Lack of Resources & Rate Limiting

APIs usually do not impose restrictions on the number or size of resources a client or user is allowed to request. However, this issue can impact the performance of the API server and result in Denial of Service (DoS). Additionally, it can create authentication flaws that enable brute force attacks.

Broken Function Level Authorization

Authorization flaws enable attackers to gain unauthorized access to the resources of legitimate users or obtain administrative privileges. It can occur as a result of overly complex access control policies based on different hierarchies, roles, groups, and unclear separation between regular and administrative functions.

Mass Assignment

Mass assignment is usually a result of improperly binding data provided by clients, like JSON, to data models. It occurs when binding happens without using properties filtering based on an allowlist. It enables attackers to guess object properties, read the documentation, explore other API endpoints, or provide additional object properties to request payloads.

Security Misconfiguration

Security misconfiguration usually occurs due to:

- Insecure default configurations
- Open cloud storage
- Ad-hoc or incomplete configurations
- Misconfigured HTTP headers
- Permissive cross-origin resource sharing (CORS)
- Unnecessary HTTP methods
- Verbose error messages that contain sensitive information

Injection

Injection flaws like command injection, SQL, and NoSQL injection occur when a query or command sends untrusted data to an interpreter. It is typically malicious data that attempts to trick the interpreter into providing unauthorized access to data or executing unintended commands.

Improper Assets Management

APIs usually expose more endpoints than traditional web applications. This nature of APIs means proper and updated documentation becomes critical to security. Additionally, proper hosts and deployed API versions inventory can help mitigate issues related to exposed debug endpoints and deprecated API versions.

Insufficient Logging & Monitoring

Insufficient logging and monitoring enable threat actors to escalate their attacks, especially when there is ineffective or no integration with incident response. It allows malicious actors to maintain persistence and pivot to other systems where they extract, destroy, or tamper with data.

What is Application Security Testing?

Application Security Testing (AST) is the process of making applications more resilient to security threats by identifying and remediating security vulnerabilities.

Originally, AST was a manual process. In modern, high-velocity development processes, AST must be automated. The increased modularity of enterprise software, numerous open source components, and a large number of known vulnerabilities and threat vectors all make automation essential. Most organizations use a combination of application security tools to conduct AST.

Key considerations before testing applications

Here are key considerations before you can properly test applications for security vulnerabilities:

- Create a complete inventory of your applications.
- Understand the business use, impact and sensitivity of your applications.
- Determine which applications to test—start from public-facing systems like web and mobile applications.

5 Recomendation

How Test

You must determine the following parameters before you can successfully test applications for security vulnerabilities:

- Authenticated vs. non-authenticated testing—you can test applications from an outsider's perspective (a black box approach). However, there is a lot of value in performing authenticated testing, to discover security issues that affect authenticated users. This can help uncover vulnerabilities like SQL injection and session manipulation.
- Which tools to use—testing should ideally involve tools that can identify vulnerabilities in source code, tools that can test applications for security weaknesses at runtime, and network vulnerability scanners.
- Testing production vs. staging—testing in production is important because it can identify security issues that are currently threatening the organization and its customers. However, production testing can have a performance impact. Testing in staging is easier to achieve and allows faster remediation of vulnerabilities.
- Whether to disable security systems while testing—for most security tests, it is a good idea to disable firewalls, web application firewalls (WAF), and intrusion prevention systems (IPS), or at least whitelist the IPs of testing tools, otherwise tools can interfere with scanning. However, in a full penetration test, tools should be left on and the goal is to scan applications while avoiding detection.
- When to test—it is typically advisable to perform security testing during off periods to avoid an impact on performance and reliability of production applications.
- What to report—many security tools provide highly detailed reports relating to their specific testing domain, and these reports are not consumable by non-security experts. Security teams should extract the most relevant insights from automated reports and present them in a meaningful way to stakeholders.

- Validation testing—a critical part of security testing is to validate that remediations were done successfully. It is not enough for a developer to say the remediation is fixed. You must rerun the test and ensure that the vulnerability no longer exists, or otherwise give feedback to developers.

Learn more in the detailed guides to:

- Cross Site Scripting (XSS)
- Local file injection (LFI)
- SQL injection (SQLi)
- Cross Site Request Forgery (CSRF)

See also

- ABX test
- Acceptance testing
- Blind experiment
- Boundary testing
- Fuzz testing
- Gray box testing
- Metasploit Project
- Sanity testing
- Smoke testing
- Software performance testing
- Software testing
- Stress testing

- Test automation
- Unit testing
- Web application security scanner
- White hat hacker
- White-box testing