

# Snyk OWASP Top 10 (2021) Report

Report Executed by **Didik Achmadi**

Report Executed on **2024-02-26**

Report Executed for **Didik-Test-1**

Analysis Includes: **Snyk Code**

Additional Scope Filters:

**Project Name**

didik-snyk/juice-shop-16(main)

---

# Report Contents

<b>Section 1:</b>	Report Status
<b>Section 2:</b>	About OWASP Top 10 (2021)
<b>Section 3:</b>	Snyk's Adherence to OWASP Top 10
<b>Section 4:</b>	Issue Summary
<b>Section 5:</b>	Issue Detail
<b>Section 6:</b>	Snyk Methodology

## Section 1: Report Status

OWASP Top 10 Web Application Security Vulnerabilities: **Pass**

Report Executed by **Didik Achmadi** on **2024-02-26**

Analysis Includes: **Snyk Code**

## Section 2: About OWASP Top 10

From <https://owasp.org/Top10/>:

The OWASP Top 10 is a standard awareness document for developers and web application security. It represents a broad consensus about the most critical security risks to web applications.

Companies should adopt this document and start the process of ensuring that their web applications minimize these risks. Using the OWASP Top 10 is perhaps the most effective first step towards changing the software development culture within your organization into one that produces more secure code.

### Top 10 Web Application Security Risks for 2021

- **A01:2021-Broken Access Control** Access control enforces policy such that users cannot act outside of their intended permissions. Failures typically lead to unauthorized information disclosure, modification, or destruction of all data or performing a business function outside the user's limits.
- **A02:2021-Cryptographic Failures** The first thing is to determine the protection needs of data in transit and at rest. For example, passwords, credit card numbers, health records, personal information, and business secrets require extra protection, mainly if that data falls under privacy laws, e.g., EU's General Data Protection Regulation (GDPR), or regulations, e.g., financial data protection such as PCI Data Security Standard (PCI DSS).
- **A03:2021-Injection** An application is vulnerable to attack when: User-supplied data is not validated, filtered, or sanitized by the application. Dynamic queries or non-parameterized calls without context-aware escaping are used directly in the interpreter. Hostile data is used within object-relational mapping (ORM) search parameters to extract additional, sensitive records. Hostile data is directly used or concatenated. The SQL or command contains the structure and malicious data in dynamic queries, commands, or stored procedures. Some of the more common injections are SQL, NoSQL, OS command, Object Relational Mapping (ORM), LDAP, and Expression Language (EL) or Object Graph Navigation Library (OGNL) injection. The concept is identical among all interpreters. Source code review is the best method of detecting if applications are vulnerable to injections. Automated testing of all parameters, headers, URL, cookies, JSON, SOAP, and XML data inputs is strongly encouraged. Organizations can include static (SAST), dynamic (DAST), and interactive (IAST) application security testing tools into the CI/CD pipeline to identify introduced injection flaws before production deployment.
- **A04:2021-Insecure Design** Insecure design is a broad category representing different weaknesses, expressed as "missing or ineffective control design." Insecure design is not the source for all other Top 10 risk categories. There is a difference between insecure design and insecure implementation. We differentiate between design flaws and implementation defects for a reason, they have different root causes and remediation. A secure design can still have implementation defects leading to vulnerabilities that may be exploited. An insecure design cannot be fixed by a perfect implementation as by definition, needed security controls were never created to defend against specific attacks. One of the factors that contribute to insecure design is the lack of business risk profiling inherent in the software or system being developed, and thus the failure to determine what level of security design is required.
- **A05:2021-Security Misconfiguration** The application might be vulnerable if the application is: Missing appropriate security hardening across any part of the application stack or improperly configured permissions on cloud services. Unnecessary features are enabled or installed (e.g., unnecessary ports, services, pages, accounts, or privileges). Default accounts and their passwords are still enabled and unchanged. Error handling reveals stack traces or other overly informative error messages to users. For upgraded systems, the latest security features are disabled or not configured securely. The security settings in the application servers, application frameworks (e.g., Struts, Spring, ASP.NET), libraries, databases, etc., are not set to secure values. The server does not send security headers or directives, or they are not set to secure values. The software is out of

date or vulnerable (see A06:2021-Vulnerable and Outdated Components). Without a concerted, repeatable application security configuration process, systems are at a higher risk.

- **A06:2021-Vulnerable and Outdated Components** You are likely vulnerable: If you do not know the versions of all components you use (both client-side and server-side). This includes components you directly use as well as nested dependencies. If the software is vulnerable, unsupported, or out of date. This includes the OS, web/application server, database management system (DBMS), applications, APIs and all components, runtime environments, and libraries. If you do not scan for vulnerabilities regularly and subscribe to security bulletins related to the components you use. If you do not fix or upgrade the underlying platform, frameworks, and dependencies in a risk-based, timely fashion. This commonly happens in environments when patching is a monthly or quarterly task under change control, leaving organizations open to days or months of unnecessary exposure to fixed vulnerabilities. If software developers do not test the compatibility of updated, upgraded, or patched libraries. If you do not secure the components' configurations (see A05:2021-Security Misconfiguration).
- **A07:2021-Identification and Authentication Failures** Confirmation of the user's identity, authentication, and session management is critical to protect against authentication-related attacks. There may be authentication weaknesses if the application: Permits automated attacks such as credential stuffing, where the attacker has a list of valid usernames and passwords. Permits brute force or other automated attacks. Permits default, weak, or well-known passwords, such as "Password1" or "admin/admin". Uses weak or ineffective credential recovery and forgot-password processes, such as "knowledge-based answers," which cannot be made safe. Uses plain text, encrypted, or weakly hashed passwords data stores (see A02:2021-Cryptographic Failures). Has missing or ineffective multi-factor authentication. Exposes session identifier in the URL. Reuse session identifier after successful login. Does not correctly invalidate Session IDs. User sessions or authentication tokens (mainly single sign-on (SSO) tokens) aren't properly invalidated during logout or a period of inactivity.
- **A08:2021-Software and Data Integrity Failures** Software and data integrity failures relate to code and infrastructure that does not protect against integrity violations. An example of this is where an application relies upon plugins, libraries, or modules from untrusted sources, repositories, and content delivery networks (CDNs). An insecure CI/CD pipeline can introduce the potential for unauthorized access, malicious code, or system compromise. Lastly, many applications now include auto-update functionality, where updates are downloaded without sufficient integrity verification and applied to the previously trusted application. Attackers could potentially upload their own updates to be distributed and run on all installations. Another example is where objects or data are encoded or serialized into a structure that an attacker can see and modify is vulnerable to insecure deserialization.
- **A09:2021-Security Logging and Monitoring Failures** This category is to help detect, escalate, and respond to active breaches. Without logging and monitoring, breaches cannot be detected. Insufficient logging, detection, monitoring, and active response occurs any time: Auditable events, such as logins, failed logins, and high-value transactions, are not logged. Warnings and errors generate no, inadequate, or unclear log messages. Logs of applications and APIs are not monitored for suspicious activity. Logs are only stored locally. Appropriate alerting thresholds and response escalation processes are not in place or effective. Penetration testing and scans by dynamic application security testing (DAST) tools (such as OWASP ZAP) do not trigger alerts. The application cannot detect, escalate, or alert for active attacks in real-time or near real-time. You are vulnerable to information leakage by making logging and alerting events visible to a user or an attacker (see A01:2021-Broken Access Control).
- **A10:2021-Server-Side Request Forgery** SSRF flaws occur whenever a web application is fetching a remote resource without validating the user-supplied URL. It allows an attacker to coerce the application to send a crafted request to an unexpected destination, even when protected by a firewall, VPN, or another type of network access control list (ACL). As modern web applications provide end-users with convenient features, fetching a URL becomes a common scenario. As a result, the incidence of SSRF is increasing. Also, the severity of SSRF is becoming higher due to cloud services and the complexity of architectures.

Visit <https://owasp.org/www-project-top-ten/> for more information.

## Section 3: Snyk's Adherence to OWASP Top 10

Snyk's products detect issues in customer applications. Customers who want to align with the OWASP Top 10 framework for managing their security issues can leverage this report to determine their compliance.

Snyk will produce a "**Pass**" status if no critical or high-severity issues map to the OWASP Top 10 controls at the time of the report generation according to the scope selected. Snyk will produce a "**Did not pass**" status if critical or high-severity issues exist.

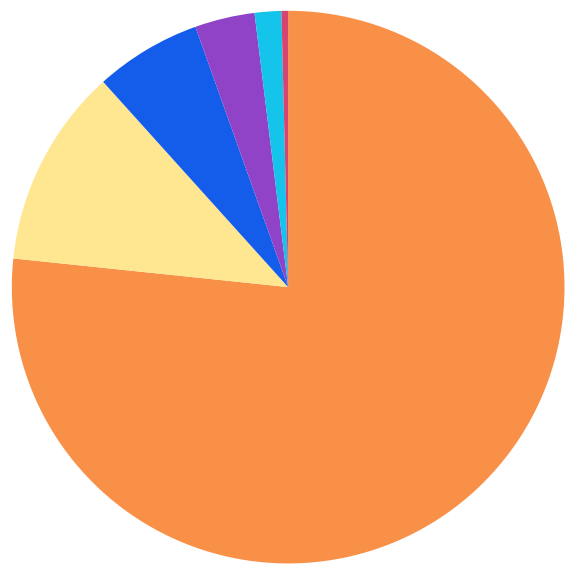
The report will reference any medium or low-severity issues found in Section 4: Issue Summary, but these issues will not result in a status of "Did not pass." Medium or low severity issues can be found in the Snyk app and will not be included in Section 5: Issue Detail section.

The mapping at Snyk between the OWASP Top 10 controls to the issues is done using CWE - Common Weakness Enumerations. Vulnerabilities at Snyk are mapped to CWE, and each OWASP control is mapped to a list of CWEs. For example, A01:Broken Access Control is mapped to 34 unique CWEs.

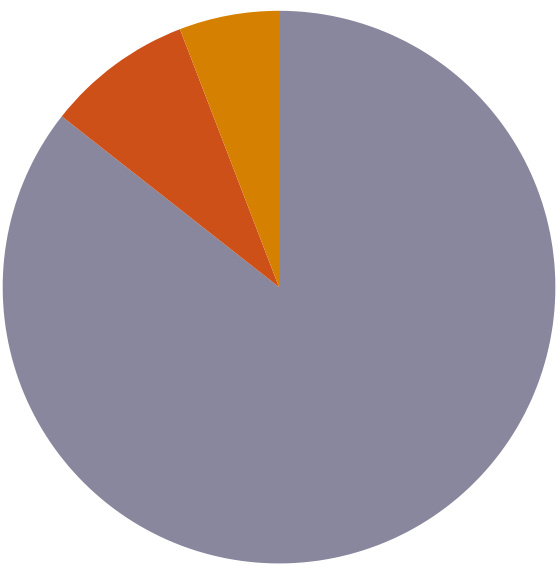
To learn more about OWASP TOP 10 security risk, and to fix and prevent them, Snyk recommend taking the OWASP TOP 10 Interactive learning path by Snyk Learn

## Section 4: Issue Summary

Distribution of Controls



Issues by Severity



### Control details

CONTROL	● CRITICAL	● HIGH	● MEDIUM	● LOW	TOTAL
● A01 Broken Access Control	0	8	1	0	9
● A02 Cryptographic Failures	0	0	1	3	4
● A03 Injection	0	8	8	0	16
● A04 Insecure Design	0	0	0	0	0
● A05 Security Misconfiguration	0	5	0	25	30
● A06 Vulnerable and Outdated Components	0	0	0	0	0
● A07 Identification and Authentication Failures	0	0	5	192	197
● A08 Software and Data Integrity Failures	0	0	0	0	0
● A09 Security Logging and Monitoring Failures	0	0	0	0	0
● A10 Server-Side Request Forgery	0	1	0	0	1
All Controls	0	22	15	220	257

## Section 5: Issue Detail

### A01 Broken Access Control

**Critical severity issues: 0**

**High severity issues: 8**

#### CWE-22 Broken Access Control

ISSUE #	ORGANIZATION	PROJECT TARGET	TARGET REFERENCE	PRODUCT
1	Didik-Test-1	didik-snyk/juice-shop-16	main	Snyk Code

#### CWE-23 Broken Access Control

ISSUE #	ORGANIZATION	PROJECT TARGET	TARGET REFERENCE	PRODUCT
1	Didik-Test-1	didik-snyk/juice-shop-16	main	Snyk Code
2	Didik-Test-1	didik-snyk/juice-shop-16	main	Snyk Code
3	Didik-Test-1	didik-snyk/juice-shop-16	main	Snyk Code
4	Didik-Test-1	didik-snyk/juice-shop-16	main	Snyk Code
5	Didik-Test-1	didik-snyk/juice-shop-16	main	Snyk Code
6	Didik-Test-1	didik-snyk/juice-shop-16	main	Snyk Code
7	Didik-Test-1	didik-snyk/juice-shop-16	main	Snyk Code

Learn more about Broken Access Control vulnerabilities.

### A03 Injection

**Critical severity issues: 0**

**High severity issues: 8**

#### CWE-79 Injection

ISSUE #	ORGANIZATION	PROJECT TARGET	TARGET REFERENCE	PRODUCT
1	Didik-Test-1	didik-snyk/juice-shop-16	main	Snyk Code
2	Didik-Test-1	didik-snyk/juice-shop-16	main	Snyk Code
3	Didik-Test-1	didik-snyk/juice-shop-16	main	Snyk Code

## CWE-89 Injection

ISSUE #	ORGANIZATION	PROJECT TARGET	TARGET REFERENCE	PRODUCT
1	Didik-Test-1	didik-snyk/juice-shop-16	main	Snyk Code
2	Didik-Test-1	didik-snyk/juice-shop-16	main	Snyk Code
3	Didik-Test-1	didik-snyk/juice-shop-16	main	Snyk Code
4	Didik-Test-1	didik-snyk/juice-shop-16	main	Snyk Code
5	Didik-Test-1	didik-snyk/juice-shop-16	main	Snyk Code

Learn more about Code Injections vulnerabilities.

## A05 Security Misconfiguration

**Critical severity issues: 0**

**High severity issues: 5**

### CWE-547 Security Misconfiguration

ISSUE #	ORGANIZATION	PROJECT TARGET	TARGET REFERENCE	PRODUCT
1	Didik-Test-1	didik-snyk/juice-shop-16	main	Snyk Code
2	Didik-Test-1	didik-snyk/juice-shop-16	main	Snyk Code
3	Didik-Test-1	didik-snyk/juice-shop-16	main	Snyk Code
4	Didik-Test-1	didik-snyk/juice-shop-16	main	Snyk Code
5	Didik-Test-1	didik-snyk/juice-shop-16	main	Snyk Code

Learn more about Security Misconfiguration vulnerabilities.

## A10 Server-Side Request Forgery

**Critical severity issues: 0**

**High severity issues: 1**

### CWE-918 Server-Side Request Forgery

ISSUE #	ORGANIZATION	PROJECT TARGET	TARGET REFERENCE	PRODUCT
1	Didik-Test-1	didik-snyk/juice-shop-16	main	Snyk Code

Learn more about SSRF (Server-Side Request Forgery) vulnerabilities.





## Section 6: Snyk Methodology

### What is Snyk?

Snyk is a developer security platform that enables application and cloud developers to secure their whole application — finding and fixing vulnerabilities from their first lines of code to their running cloud.





### Industry-leading security intelligence

Snyk security researchers augment their expertise with advanced ML and human-in-the-loop AI so we can provide the most accurate, timely and comprehensive intelligence on the market. This security intel is the foundation of our platform, spanning the Snyk Vulnerability Database, the Snyk Code knowledge base, and our Cloud/IaC unified policy engine.

### Snyk Severity Levels

A severity level is applied to a vulnerability, to indicate the risk for that vulnerability in an application.





Severity levels are key factors in vulnerability assessment, and can be:

Severity	Severity Level	Description
	Critical	This may allow attackers to access sensitive data and run code on your application
	High	This may allow attackers to access sensitive data in your application
	Medium	Under some conditions, this may allow attackers to access sensitive data on your application
	Low	Application may expose some data that allows vulnerability mapping, which can be used with other vulnerabilities to attack the application

### Determining severity levels - Snyk Open Source and Snyk Container

The Common Vulnerability Scoring System (CVSS) determines the severity level of a vulnerability.

Snyk uses CVSS framework version 3.1 to communicate the characteristics and severity of vulnerabilities.

Severity	Severity Level	CVSS score
	Low	0.0 - 3.9
	Medium	4.0 - 6.9
	High	7.0 - 8.9
	Critical	9.0 - 10.0

The severity level and score are determined based on the CVSS Base Score calculations using the Base Metrics.

See Scoring security vulnerabilities 101: Introducing CVSS for CVEs to learn more.

Severity levels may not always align with CVSS scores. For example, Snyk Container severity scores for Linux vulnerabilities may vary depending on NVD severity rankings; see Understanding Linux vulnerability severity for more details.

For more information about CVSS and severity levels please see the following documentation.