OWASP Top 10:

A01. Broken Access Control

If authentication and access restriction are not properly implemented, it's easy for attackers to take whatever they want. With broken access control flaws, unauthenticated or unauthorized users may have access to sensitive files and systems, or even user privilege settings. Penetration testing can detect missing authentication but cannot determine the misconfigurations that lead to the exposure. One of the benefits of the increasing use of Infrastructure as Code (IaC) tools is the ability to use scanning tools to detect configuration errors leading to access control failures. Weak access controls and issues with credentials management in applications are preventable with secure coding practices, as well as preventative measures like locking down administrative accounts and controls and using multi-factor authentication.

Example:

Session Fixation: An attacker can force a user's session ID to a known value and then wait for the user to authenticate using that session ID. Once the user logs in, the attacker can use the fixed session ID to gain unauthorized access.

A02: Cryptographic Failures

Common errors such as using hardcoded passwords, outdated cryptographic algorithms, or weak cryptographic keys can result in the exposure of sensitive data. Scanning images for hardcoded secrets, and ensuring that data is properly encrypted at rest and in transit can help mitigate exposing sensitive data to attackers.

Example:

Cryptographic Side-Channel Attacks: An application is vulnerable to side-channel attacks, such as timing attacks or power analysis attacks, which exploit the physical implementation of cryptographic algorithms to extract sensitive information. Attackers can exploit these weaknesses to recover cryptographic keys or plaintext data by analyzing the timing or power consumption of cryptographic operations.

A03: Injection

Injection attacks occur when attackers exploit vulnerabilities in web applications that accept untrusted data. Common types include SQL injection and OS command injection. This category now also includes Cross Site Scripting (XSS). By inserting malicious code into input fields, attackers can execute unauthorized commands, access sensitive databases, and potentially gain control over systems.Application security testing can reveal injection flaws and suggest remediation techniques such as stripping special characters from user input or writing parameterized SQL queries.

Example:

```
username = getRequestParameter("username")

password = getRequestParameter("password")

sql_query = "SELECT * FROM users WHERE username='" + username + "' AND password='" + password
+ "'"

executeSQLQuery(sql_query)
```

An attacker can exploit this vulnerability by manipulating the input fields to inject malicious SQL code. For example, entering ' OR '1'='1' as the username and leaving the password field empty results in the following SQL query:

SELECT * FROM users WHERE username='' OR '1'='1' AND password=''

Since the condition '1'='1' always evaluates to true, the query returns all rows from the users table, effectively bypassing authentication and granting unauthorized access to the application. To mitigate SQL injection vulnerabilities, developers should use parameterized queries or prepared statements, which separate the SQL code from user input and automatically handle escaping special characters, preventing attackers from injecting malicious code into queries.

A04: Insecure Design

Insecure design is a new category in the 2021 OWASP Top Ten which focusses on fundamental design flaws and ineffective controls as opposed to weak or flawed implementations. Creating secure designs and secure software development lifecycles requires a combination of culture, methodologies and tools. Developer training, robust threat modelling, and an organizational library of secure design patterns should all be implemented to reduce the risks of insecure designs creating critical vulnerabilities.

Example:

Serialized Object Tampering: An attacker modifies the serialized form of an object to include malicious code or change its properties. When the object is deserialized by the application, the attacker's code is executed, allowing them to take control of the application.

A05: Security Misconfiguration

Application servers, frameworks, and cloud infrastructure are highly configurable, and security misconfigurations such as too broad permissions, insecure default values left unchanged, or too revealing error messages can provide attackers easy paths to compromise applications. The 2023 Veracode State of Software Security reported that misconfiguration errors were reported in 70% or more applications that had introduced a new vulnerability in the last year.To reduce misconfiguration risks organizations should routinely harden deployed application and infrastructure configurations and should scan all infrastructure as code components as part of a secure SDLC.

Example:

Default Credentials: Failure to change default credentials, such as usernames and passwords for administrative interfaces, leaves the system vulnerable to unauthorized access. An attacker can easily locate default credentials online and use them to gain control over the system.

A06: Vulnerable and Outdated Components

Modern applications are built using a large number of third-party libraries (which themselves are dependent on other libraries), and frequently run on open-source frameworks. In a modern application there may be orders of magnitude more code from libraries and components than written by an organization’s developers. As might be expected with any software, vulnerabilities in libraries and components will routinely be discovered, patched, and new versions released. The challenges of identifying all the components in use, keeping track of their vulnerability status, and routinely rebuilding and testing deployed software is both essential and onerous. Perhaps this is why so many organizations are still running vulnerable software in production. A critical mitigation step is to build a Software Bill of Materials (SBoM) for all the software deployed or supplied to customers. Veracode Software Composition analysis and Container Scanning tools can produce SBoMs in standardized formats to give organizations a view of their exposure to vulnerabilities in third-party components.

Example:

Outdated Library: An application uses an outdated version of a JavaScript library known to have a critical security vulnerability. Attackers can exploit this vulnerability to execute arbitrary code on the server or steal sensitive data from users.

A07: Identification and Authentication Failures

Identifying and authorizing users and non-human clients is a fundamental security practice. It goes without saying that weaknesses in a way an application allows access or identifies users is a critical vulnerability. While mitigation starts with secure coding practices, tools to detect and prevent credential stuffing and brute force attacks are also useful protections.

Example:

Weak Password Policies: In a system with weak password policies, users are allowed to choose passwords that are easy to guess or crack, making it easier for attackers to gain unauthorized access.For instance, consider a system that allows users to set passwords without enforcing any complexity requirements. A user sets their password as "password123". This weak password is easily guessed or cracked by attackers using common password cracking techniques, such as dictionary attacks or brute force attacks. Once the attacker obtains the user's password, they can access the system and potentially compromise sensitive data or perform malicious actions.To mitigate this vulnerability, organizations should implement strong password policies that require users to create complex passwords containing a mix of uppercase and lowercase letters, numbers, and special characters. Additionally, enforcing password expiration and implementing multi-factor authentication (MFA) can provide an extra layer of security.

A08: Software and Data Integrity Failures

The tools used to build, manage, or deploy software are increasingly common vectors of attack. A CI'CD pipeline that routinely builds, tests and deploys software can also be used to inject malicious code (or libraires), create insecure deployments, or steal secrets. 'Vulnerable and Outdated Components' modern applications use many third-party components, often pulling them from third party repositories. Organizations can mitigate this threat by ensuring both the security of the build process, and the components pulled into the build. Adding in code scanning and software component analysis steps into a software build pipeline can identify malicious code or libraries.

Example:

Data Tampering: Data tampering occurs when attackers modify or manipulate data in transit or at rest, leading to unauthorized changes or corruption of the data.For example, consider an e-commerce website that stores customer order information in a database. If the application fails to properly authenticate and authorize requests to modify order details, attackers can intercept and modify the requests to change the quantity or price of items in an order before it is processed. This could result in financial loss for the e-commerce website or its customers.To prevent data tampering, organizations should implement strong authentication and authorization mechanisms to control access to sensitive data and transactions. Additionally, employing encryption techniques can help protect data both in transit and at rest, ensuring its integrity and confidentiality.

A09: Security Logging and Monitoring Failures

Having adequate logging and monitoring in place is essential in both detecting a breach early, hopefully limiting the damage, and in incident forensics to establish the scope of the breach, and to determine the method of compromise. Simply generating the data is obviously insufficient, organizations must have adequate collection, storage, alerting and escalation processes. Organizations should also verify that these processes are working correctly using Dynamic Application Security Testing (DAST) tools like Veracode DAST, for instance, should produce significant logging and alerting events.

Example:

Lack of Log Analysis: An attacker gains unauthorized access to a web server and exfiltrates sensitive data. However, the server's logs are not adequately monitored, and the attack goes unnoticed until it's too late. With proper log analysis and monitoring, the intrusion could have been detected and mitigated in a timely manner.

A10: Server-Side Request Forgery (SSRF)

Modern web applications commonly fetch additional content or data from a remote resource. If an attacker can influence the destination resource, and the application does not validate the supplied URL, then a crafted request may be sent to a target destination. Mitigating SSRF attacks is done using familiar methods such as sanitizing user input, using explicit allow lists, and inspecting request responses before they are returned to clients.

Example:

```
import requests

def fetch_url_content(url):

    response = requests.get(url)

    return response.text

user_url = getRequestParameter("url")

content = fetch_url_content(user_url)

display_content(content)
```

The application fetches the content of the provided URL using the requests.get() function from the popular Requests library in Python. However, the application fails to properly validate and sanitize the user-supplied URL.An attacker exploits this vulnerability by providing a malicious URL pointing to an internal resource rather than an external webpage:

http://localhost/admin

In this example, the attacker's intent is to access the admin interface of the server, which is typically restricted to internal network access only. When the application makes a request to the supplied URL (http://localhost/admin), it bypasses the intended functionality and accesses the internal admin interface instead. The attacker can exploit this access to potentially gain unauthorized control over the server or retrieve sensitive information from the admin interface.