# OWASP

The Open Web Application Security Project (OWASP) is a nonprofit foundation dedicated to improving software security. It operates under an “open community” model, which means that anyone can participate in and contribute to OWASP-related online chats, projects, and more. For everything from online tools and videos to forums and events, the OWASP ensures that its offerings remain free and easily accessible through its website.

The OWASP Top 10 provides rankings of—and remediation guidance for—the top 10 most critical web application security risks. Leveraging the extensive knowledge and experience of the OWASP’s open community contributors, the report is based on a consensus among security experts from around the world. Risks are ranked according to the frequency of discovered security defects, the severity of the uncovered vulnerabilities, and the magnitude of their potential impacts. The purpose of the report is to offer developers and web application security professionals insight into the most prevalent security risks so that they may fold the report’s findings and recommendations into their own security practices, thereby minimizing the presence of known risks in their applications.

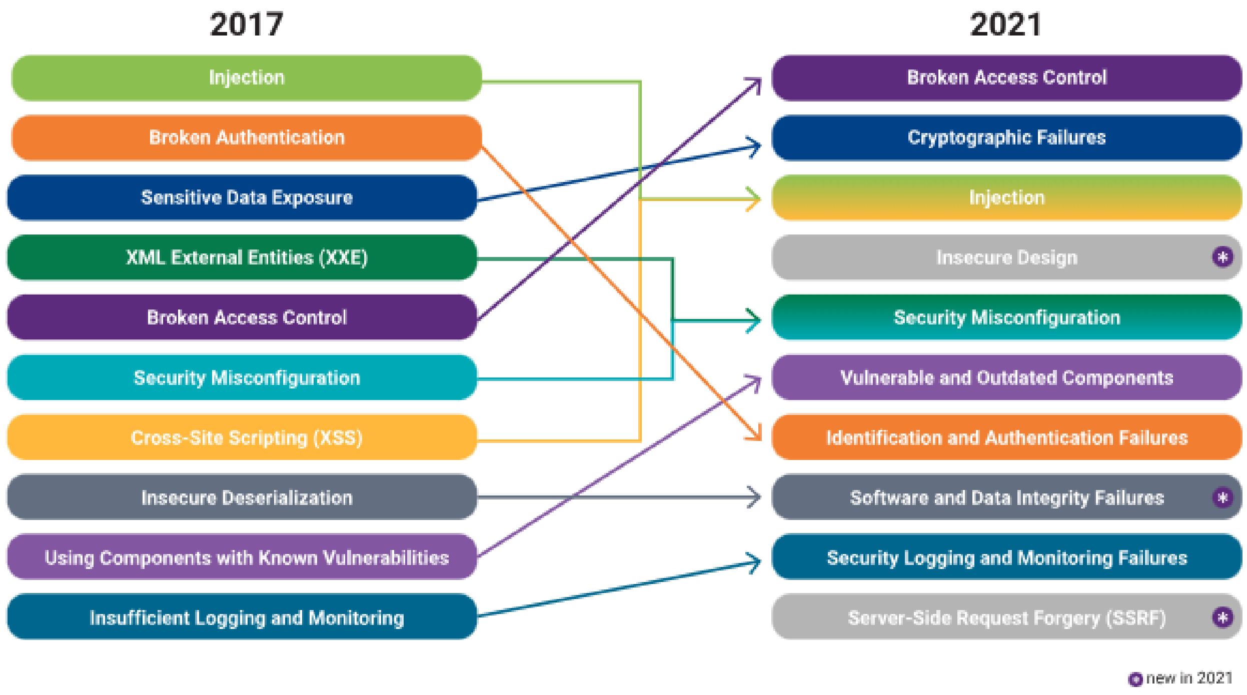
## How is the OWASP Top 10 list used and why is it important?

The OWASP has maintained its Top 10 list since 2003, updating it every two or three years in accordance with advancements and changes in the AppSec market. The list’s importance lies in the actionable information it provides in serving as a checklist and internal web application development standard for many of the world’s largest organizations.

Auditors often view an organization’s failure to address the OWASP Top 10 as an indication that it may be falling short on other compliance standards. Conversely, integrating the Top 10 into the software development life cycle (SDLC) demonstrates an organization’s overall commitment to industry best practices for secure development.

## What’s new in the 2021 list?

For the 2021 list, the OWASP added three new categories, made four changes to naming and scoping, and did some consolidation.


2017
Injection
Broken Authentication
Sensitive Data Exposure
XML External Entities (XXE)
Broken Access Control
Security Misconfiguration
Cross-Site Scripting (XSS)
Insecure Deserialization
Using Components with Known Vulnerabilities
Insufficient Logging and Monitoring
2021
Broken Access Control
Cryptographic Failures
Injection
Insecure Design
Security Misconfiguration
Vulnerable and Outdated Components
Identification and Authentication Failures
Software and Data Integrity Failures
Security Logging and Monitoring Failures
Server-Side Request Forgery (SSRF)
new in 2021

## 1. Broken Access Control (A01:2021)

Previously number 5 on the list, broken access control—a weakness that allows an attacker to gain access to user accounts—moved to number 1 for 2021. The attacker in this context can function as a user or as an administrator in the system.

**Example:** An application allows a primary key to be changed, and when this key is changed to another user's record, that user's account can be viewed or modified.

**Solution:** An interactive application security testing (IAST) solution, such as Seeker®, can help you effortlessly detect cross-site request forgery or insecure storage of your sensitive data. It also pinpoints any bad or missing logic being used to handle JSON Web Tokens. Penetration testing can serve as a manual supplement to IAST activities, helping to detect unintended access controls. Changes in architecture and design may be warranted to create trust boundaries for data access.

## 2. Cryptographic Failures (A02:2021).

Previously in position number 3 and formerly known as sensitive data exposure, this entry was renamed as cryptographic failures to accurately portray it as a root cause, rather than a symptom. Cryptographic failures occur when important stored or transmitted data (such as a social security number) is compromised.

**Example:** A financial institution fails to adequately protect its sensitive data and becomes an easy target for credit card fraud and identity theft.

**Solution:** Seeker's checkers can scan for both inadequate encryption strength and weak or hardcoded cryptographic keys, and then identify any broken or risky cryptographic algorithms. The Black Duck® cryptography module surfaces the cryptographic methods used in open source software (OSS) so they can be further evaluated for strength. Both Coverity® static application security testing (SAST) and Black Duck software composition analysis (SCA) have checkers that can provide a "point in time" snapshot at the code and component levels. However, supplementing with IAST is critical for providing continuous monitoring and verification to ensure that sensitive data isn't leaked during integrated testing with other internal and external software components.

## 3. Injection (A03:2021)

Injection moves down from number 1 to number 3, and cross-site scripting is now considered part of this category. Essentially, a code injection occurs when invalid data is sent by an attacker into a web application in order to make the application do something it was not designed to do.

**Example:** An application uses untrusted data when constructing a vulnerable SQL call.

**Solution:** Including SAST and IAST tools in your continuous integration / continuous delivery (CI/CD) pipeline helps identify injection flaws both at the static code level and dynamically during application runtime testing. Modern application security testing (AST) tools such as Seeker can help secure the software application during the various test stages and check for a variety of injection attacks (in addition to SQL injections). For example, it can identify NoSQL injections, command injections, LDAP injections, template injections, and log injections. Seeker is the first tool to provide a new, dedicated checker designed to specifically detect Log4Shell vulnerabilities, determine how Log4J is configured, test how it actually behaves, and validate (or invalidate) those findings with its patented Active Verification engine.

## 4. Insecure Design (A04:2021)

Insecure design is a new category for 2021 that focuses on risks related to design flaws. As organizations continue to "shift left," threat modeling, secure design patterns and principles, and reference architectures are not enough.

**Example:** A movie theater chain that allows group booking discounts requires a deposit for groups of more than 15 people. Attackers threat model this flow to see if they can book hundreds of seats across various theaters in the chain, thereby causing thousands of dollars in lost income.

**Solution:** Seeker IAST detects vulnerabilities and exposes all the inbound and outbound API, services, and function calls in highly complex web, cloud, and microservices-based applications. By providing a visual map of the data flow and endpoints involved, any weaknesses in the design of the app design are made clear, aiding in pen testing and threat modeling efforts.

## 5. Security Misconfiguration (A05:2021)

The former external entities category is now part of this risk category, which moves up from the number 6 spot. Security misconfigurations are design or configuration weaknesses that result from a configuration error or shortcoming.

**Example:** A default account and its original password are still enabled, making the system vulnerable to exploit.

**Solution:** Solutions like Coverity SAST include a checker that identifies the information exposure available through an error message. Dynamic tools like Seeker IAST can detect information disclosure and inappropriate HTTP header configurations during application runtime testing.

## 6. Vulnerable and Outdated Components (A06:2021)

This category moves up from number 9 and relates to components that pose both known and potential security risks, rather than just the former. Components with known vulnerabilities, such as CVEs, should be identified and patched, whereas stale or malicious components should be evaluated for viability and the risk they may introduce.

**Example:** Due to the volume of components used in development, a development team might not know or understand all the components used in their application, and some of those components might be out-of-date and therefore vulnerable to attack.

**Solution:** Software composition analysis (SCA) tools like Black Duck can be used alongside static analysis and IAST to identify and detect outdated and insecure components in an application. IAST and SCA work well together, providing insight into how vulnerable or outdated components are actually being used. Seeker IAST and Black Duck SCA together go beyond identifying a vulnerable component, uncovering details like whether that component is currently loaded by an application under test. Additionally, metrics such as developer activity, contributor reputation, and version history can give users an idea of the potential risk that a stale or malicious component may pose.

## 7. Identification and Authentication Failures (A07:2021)

Previously known as broken authentication, this entry has moved down from number 2 and now includes CWEs related to identification failures. Specifically, functions related to authentication and session management, when implemented incorrectly, allow attackers to compromise passwords, keywords, and sessions, which can lead to stolen user identity and more.

**Example:** A web application allows the use of weak or easy-to-guess passwords (i.e., “password1”).

**Solution:** Multifactor authentication can help reduce the risk of compromised accounts, and automated static analysis is highly useful in finding such flaws, while manual static analysis can add strength when evaluating custom authentication schemes. Coverity SAST includes a checker that specifically identifies broken authentication vulnerabilities. Seeker IAST can detect hardcoded passwords and credentials, as well improper authentication or missing critical steps in authentication.

## 8. Software and Data Integrity Failures (A08:2021).

This is a new category for 2021 that focuses on software updates, critical data, and CI/CD pipelines used without verifying integrity. Also now included in this entry, insecure deserialization is a deserialization flaw that allows an attacker to remotely execute code in the system.

**Example:** An application deserializes attacker-supplied hostile objects, opening itself to vulnerability.

**Solution:** Application security tools help detect deserialization flaws, and penetration testing can validate the problem. Seeker IAST can also check for unsafe deserialization and help detect insecure redirects or any tampering with token access algorithms.

## 9. Security Logging and Monitoring Failures (A09:2021)

Formerly known as insufficient logging and monitoring, this entry has moved up from number 10 and has been expanded to include more types of failures. Logging and monitoring are activities that should be performed on a website frequently—failure to do so leaves a site vulnerable to more severe compromising activities.

**Example:** Events that can be audited, like logins, failed logins, and other important activities, are not logged, leading to a vulnerable application.

**Solution:** After performing penetration testing, developers can study test logs to identify possible shortcomings and vulnerabilities. Coverity SAST and Seeker IAST can help identify unlogged security exceptions.

## 10. Server-Side Request Forgery (A10:2021)

A new category this year, a server-side request forgery (SSRF) can happen when a web application fetches a remote resource without validating the user-supplied URL. This allows an attacker to make the application send a crafted request to an unexpected destination, even when the system is protected by a firewall, VPN, or additional network access control list. The severity and incidence of SSRF attacks are increasing due to cloud services and the increased complexity of architectures.

**Example:** If a network architecture is unsegmented, attackers can use connection results or elapsed time to connect or reject SSRF payload connections to map out internal networks and determine if ports are open or closed on internal servers.

**Solution:** Seeker is one of the modern AST tools that can track, monitor, and detect SSRF without the need for additional scanning and triaging. Due to its advanced instrumentation and agent-based technology, Seeker can pick up any potential exploits from SSRF as well.

# Application Security Testing (AST)

Application security testing (AST) is the process of making applications more resistant to security threats, by identifying security weaknesses and vulnerabilities in source code.

AST started as a manual process. Today, due to the growing modularity of enterprise software, the huge number of open source components, and the large number of known vulnerabilities and threat vectors, AST must be automated. Most organizations use a combination of several application security tools.

## Static Application Security Testing (SAST)

SAST tools use a white box testing approach, in which testers inspect the inner workings of an application. SAST inspects static source code and reports on security weaknesses.

Static testing tools can be applied to non-compiled code to find issues like syntax errors, math errors, input validation issues, invalid or insecure references. They can also run on compiled code using binary and byte-code analyzers.

## Dynamic Application Security Testing (DAST)

DAST tools take a black box testing approach. They execute code and inspect it in runtime, detecting issues that may represent security vulnerabilities. This can include issues with query strings, requests and responses, the use of scripts, memory leakage, cookie and session handling, authentication, execution of third-party components, data injection, and DOM injection.

DAST tools can be used to conduct large-scale scans simulating a large number of unexpected or malicious test cases and reporting on the application’s response.

## Interactive Application Security Testing (IAST)

IAST tools are the evolution of SAST and DAST tools—combining the two approaches to detect a wider range of security weaknesses. Like DAST tools, IAST tools run dynamically and inspect software during runtime. However, they are run from within the application server, allowing them to inspect compiled source code like IAST tools do.

IAST tools can provide valuable information about the root cause of vulnerabilities and the specific lines of code that are affected, making remediation much easier. They can analyze source code, data flow, configuration and third-party libraries, and are suitable for API testing.

## Mobile Application Security Testing (MAST)

MAST tools combine static analysis, dynamic analysis and investigation of forensic data generated by mobile applications. They can test for security vulnerabilities like SAST, DAST and IAST, and in addition address mobile-specific issues like jailbreaking, malicious wifi networks, and data leakage from mobile devices.

## Software Composition Analysis (SCA)

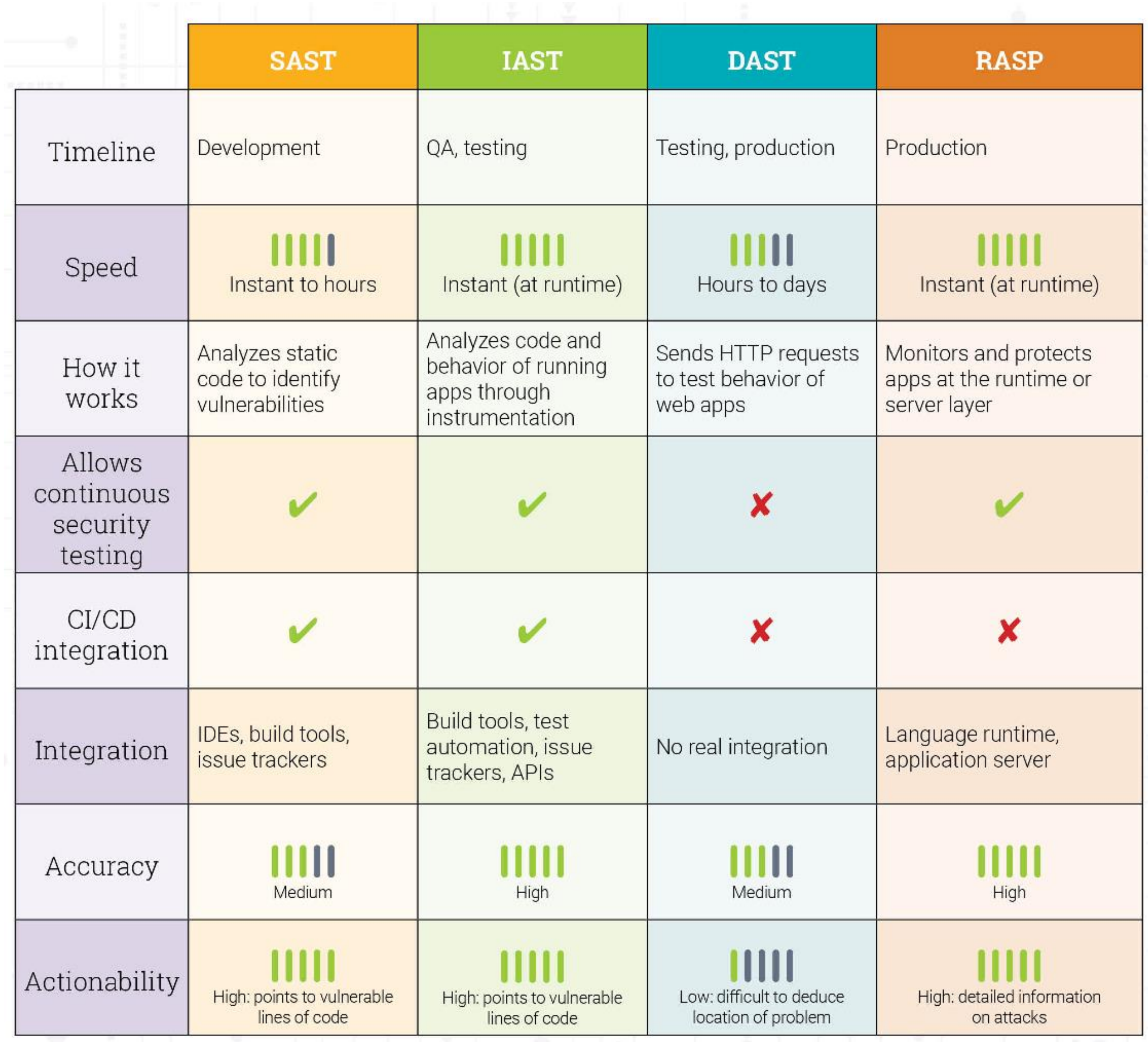
SCA tools help organizations conduct an inventory of third-party commercial and open source components used within their software. Enterprise applications can use thousands of third-party components, which may contain security vulnerabilities. SCA helps understand which components and versions are actually being used, identify the most severe security vulnerabilities affecting those components, and understand the easiest way to remediate them.

## Runtime Application Self-Protection (RASP)

RASP tools evolved from SAST, DAST and IAST. They are able to analyze application traffic and user behavior at runtime, to detect and prevent cyber threats.

Like the previous generation of tools, RASP has visibility into application source code and can analyze weaknesses and vulnerabilities. It goes one step further by identifying that security weaknesses have been exploited, and providing active protection by terminating the session or issuing an alert.

RASP tools integrate with applications and analyze traffic at runtime, and can not only detect and warn about vulnerabilities, but actually prevent attacks. Having this type of in-depth inspection and protection at runtime makes SAST, DAST and IAST much less important, making it possible to detect and prevent security issues without costly development work.



| | SAST | IAST | DAST | RASP |
|---|---|---|---|---|
| Timeline | Development | QA, testing | Testing, production | Production |
| Speed | Instant to hours | Instant (at runtime) | Hours to days | Instant (at runtime) |
| How it works | Analyzes static code to identify vulnerabilities | Analyzes code and behavior of running apps through instrumentation | Sends HTTP requests to test behavior of web apps | Monitors and protects apps at the runtime or server layer |
| Allows continuous security testing | ✓ | ✓ | ✗ | ✓ |
| CI/CD integration | ✓ | ✓ | ✗ | ✗ |
| Integration | IDEs, build tools, issue trackers | Build tools, test automation, issue trackers, APIs | No real integration | Language runtime, application server |
| Accuracy | Medium | High | Medium | High |
| Actionability | High: points to vulnerable lines of code | High: points to vulnerable lines of code | Low: difficult to deduce location of problem | High: detailed information on attacks |

## Application Security Testing Best Practices

### Shift security testing left

New organizational practices like DevSecOps are emphasizing the need to integrate security into every stage of the software development lifecycle. AST tools can:

- Help developers understand security concerns and enforce security best practices at the development stage.
- Help testers identify security issues early before software ships to production.
- Advanced tools like RASP can identify and block vulnerabilities in source code in production.

## Test internal interfaces, not just APIs and UIs

It is natural to focus application security testing on external threats, such as user inputs submitted via web forms or public API requests. However, it is even more common to see attackers exploit weak authentication or vulnerabilities on internal systems, once already inside the security perimeter. AST should be leveraged to test that inputs, connections and integrations between internal systems are secure.

## Test often

New vulnerabilities are discovered every day, and enterprise applications use thousands of components, any of which could go end of life (EOL) or require a security update. It is essential to test critical systems as often as possible, prioritize issues focusing on business critical systems and high-impact threats, and allocate resources to remediate them fast.

## Third-party code security

Organizations should employ AST practices to any third-party code they use in their applications. Never “trust” that a component from a third party, whether commercial or open source, is secure. Scan third-party code just like you scan your own. If you discover severe issues, apply patches, consult vendors, create your own fix or consider switching components.