Web Application Security

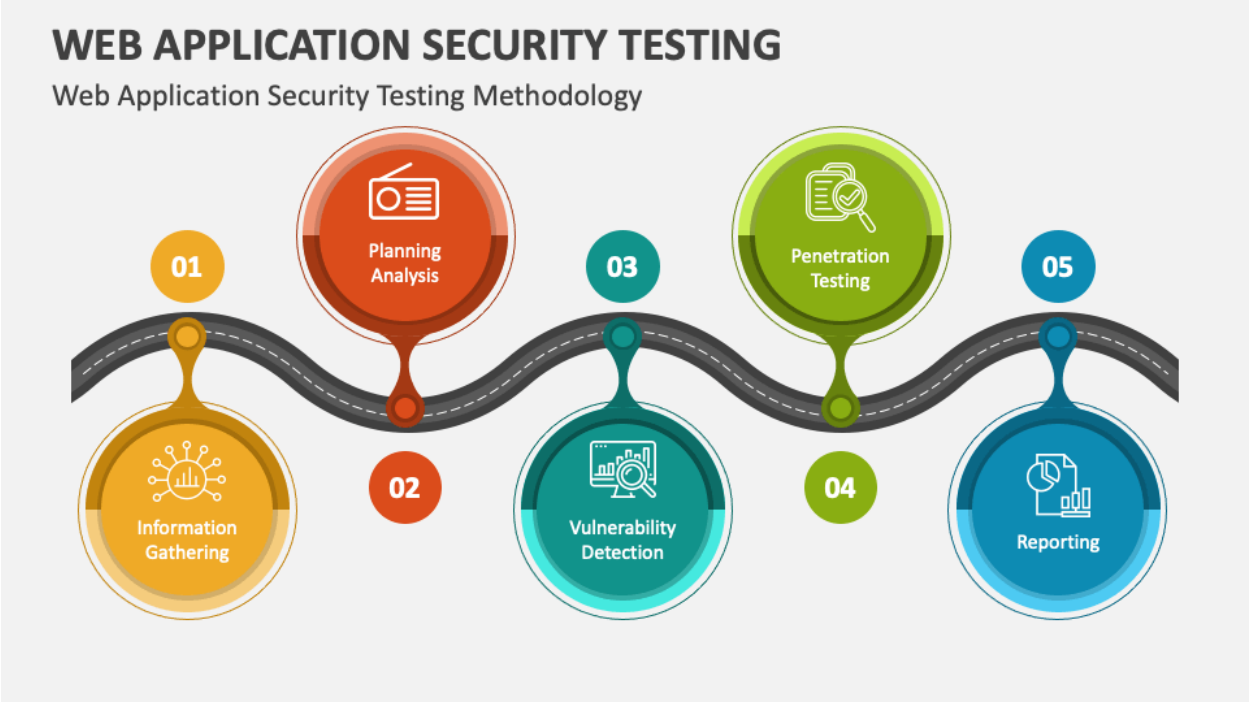
Report by

SWEATA BISWAS

Web Application Security

Testing methodologies for web application security

Here's a detailed explanation of the testing methodologies for web application security:



1. Information Gathering (Reconnaissance)

Collect as much information as possible about the target web application to understand its structure, technologies used, and potential vulnerabilities.

- **Passive Reconnaissance**: Gathering information without directly interacting with the target.
    - **DNS Information**: Tools like whois and dig to retrieve domain registration details, DNS records, etc.
    - **Google Dorking**: Use advanced search queries to find sensitive information exposed on search engines.

    - **Social Media Mining**: Gather insights from the company's social media presence that could reveal internal details.
    - **Footprinting**: Identify the server's operating system, web server, and application software versions.

- **Active Reconnaissance**: Involves direct interaction with the target system to collect information.

    - **Port Scanning**: Tools like Nmap to identify open ports and services running on the web server.
    - **Service Fingerprinting**: Identify software and versions in use (e.g., using whatweb or Wappalyzer).
    - **Subdomain Enumeration**: Discover subdomains using tools like Sublist3r or Amass.
    - **Spidering**: Crawl the web application to discover all available pages and endpoints (e.g., using tools like Burp Suite).

2. Planning and Analysis

Develop a detailed testing plan, assess the risks, and understand the application logic and scope of the test.

- **Define Testing Scope**: Identify the parts of the application to be tested (e.g., specific features, APIs, or modules).
- **Risk Assessment**: Prioritize areas based on the sensitivity of data processed, exposure to external threats, and business impact.
- **Testing Strategy**: Choose the tools and techniques that will be used in each phase of testing. This includes deciding between manual testing, automated scanning, or a combination of both.
- **Legal and Ethical Considerations**: Ensure all necessary permissions and agreements are in place before conducting any tests.
- **Test Plan Document**: Detailing scope, objectives, timeline, and methodologies.
- **Risk Profile**: A list of potential risks categorized by severity.

3. Vulnerability Detection

Identify security vulnerabilities in the web application using both automated tools and manual techniques.

- **Automated Vulnerability Scanning**:
  - **Tools**: Use OWASP ZAP, Burp Suite, Nikto, etc., to scan for common vulnerabilities like SQL injection, Cross-Site Scripting (XSS), insecure HTTP headers, etc.
  - **Output**: Automated reports that highlight potential vulnerabilities, which require further validation.

- **Manual Testing**:
  - **Authentication and Session Management**: Test login mechanisms, password recovery features, multi-factor authentication, and session handling for flaws (e.g., session fixation).
  - **Input Validation Testing**: Manually test inputs using various payloads to identify vulnerabilities like SQL injection, XSS, or command injection.
  - **Cross-Site Scripting (XSS)**: Inject JavaScript payloads into forms, URLs, or headers to check for improper input validation and output encoding.
  - **SQL Injection**: Manually test query inputs to see if it's possible to manipulate database queries. Tools like SQLMap can automate parts of this process.
  - **Insecure Direct Object References (IDOR)**: Check if access controls are properly implemented by attempting to access unauthorized resources (e.g., modifying URLs to access other users' data).
  - **File Upload Testing**: Test for the ability to upload malicious files, leading to potential Remote Code Execution (RCE).
  - **Business Logic Testing**: Test scenarios that might break the intended application logic, such as bypassing workflows or conducting unauthorized transactions.

- **Configuration and Deployment Management Testing**:

- **Check for Default Credentials**: Verify if any default usernames/passwords are being used.
- **Sensitive Information Disclosure**: Look for error messages, configuration files, or debug information that might be exposed.
- **SSL/TLS Testing**: Ensure the proper configuration of SSL/TLS (e.g., testing for weak ciphers, lack of HSTS).

4. Penetration Testing

Simulate real-world attacks to exploit identified vulnerabilities, assess their impact, and understand the extent of potential damage.

- **Exploitation**:
  - **Perform Exploits**: Based on the vulnerabilities discovered, attempt to exploit them to gain unauthorized access, extract data, or perform other malicious actions.
  - **Attack Vectors**: Exploitation methods may include SQL Injection to retrieve database information, XSS to steal cookies or session tokens, CSRF to perform unauthorized actions, etc.
  - **Tool Support**: Tools like Metasploit can help automate exploitation but should be used with caution.
- **Privilege Escalation**:
  - **Post-Exploitation**: After gaining a foothold, attempt to escalate privileges from a lower-level account to a higher one (e.g., from a regular user to an admin).
  - **Lateral Movement**: Attempt to move within the application or even to other connected systems.
- **Persistence**:
  - **Establishing Backdoors**: Simulate persistence by installing web shells or backdoors that allow continuous access even after the initial exploitation vector is patched.
- **Impact Analysis**:

- **Data Exfiltration**: Test how much data can be extracted from the application.
- **Service Disruption**: Simulate denial-of-service (DoS) attacks to assess the impact on application availability.
- **Defacement or Manipulation**: Alter the application in ways that demonstrate potential harm without causing actual damage.

5. Reporting

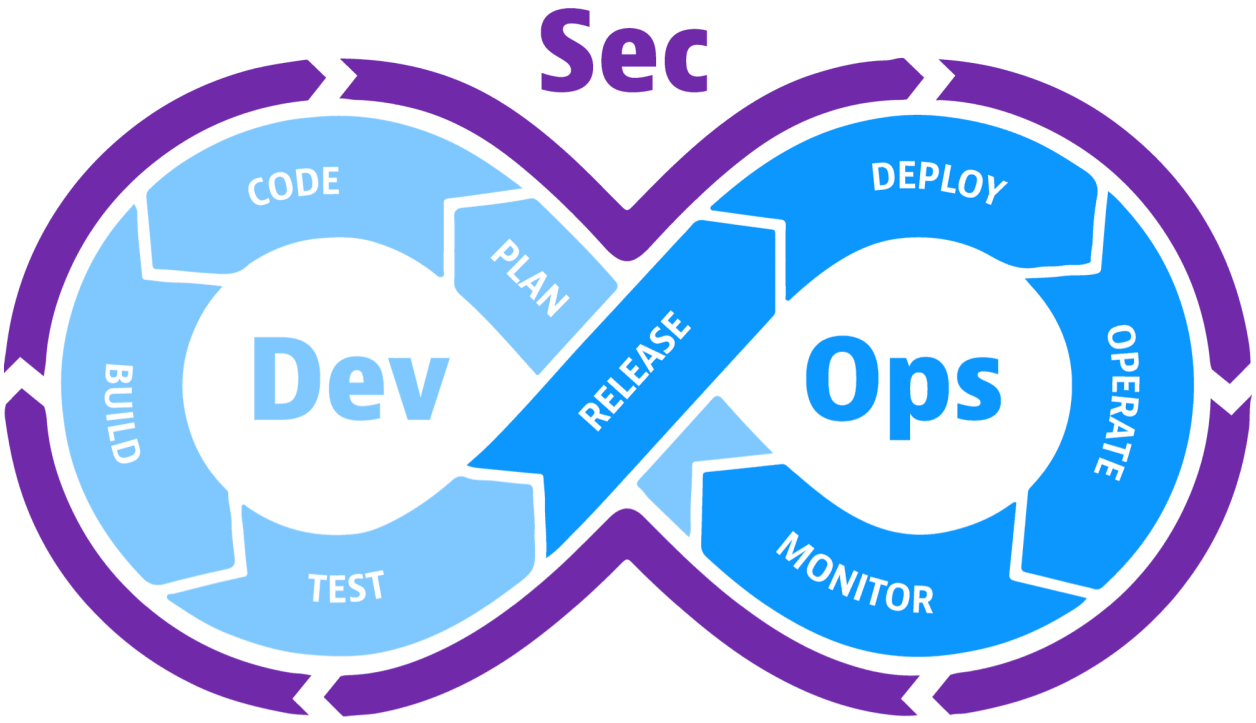
Document and present findings in a clear, actionable format for stakeholders, including developers, security teams, and management.

- **Executive Summary**:
  - **Overview**: A high-level summary of the test objectives, key findings, and overall security posture.
  - **Risk Assessment**: Highlight the most critical vulnerabilities with potential business impacts.
- **Detailed Findings**:
  - **Vulnerability Description**: A thorough explanation of each vulnerability, including its nature, affected components, and how it was discovered.
  - **Risk Level**: Assign severity levels (e.g., Low, Medium, High, Critical) based on the impact and likelihood of exploitation.
  - **Proof of Concept (PoC)**: Provide evidence of successful exploitation, including screenshots, logs, or code snippets.
  - **Affected Assets**: Detail the components or areas of the application that are impacted by each vulnerability.
- **Remediation Recommendations**:
  - **Fix Guidelines**: Provide specific steps for developers or system administrators to remediate the vulnerabilities.
  - **Best Practices**: Suggest security best practices to prevent similar vulnerabilities in the future.
  - **Mitigation Strategies**: For vulnerabilities that cannot be fully resolved, offer mitigation techniques to reduce risk.

- **Conclusion and Next Steps**:
    - **Summary**: Recap the key takeaways from the testing process.
    - **Future Actions**: Recommend re-testing after remediation or continuous monitoring solutions.
    - **Further Security Enhancements**: Suggest additional security measures, such as implementing a Web Application Firewall (WAF), regular security audits, or user training on security awareness.
- **Appendices**:
    - **Tools Used**: List all the tools used during testing.
    - **Detailed Logs**: Include logs of testing activities for reference.
    - **References**: Provide links or citations for any standards or guidelines used (e.g., OWASP Top Ten).

DevSecOps

DevSecOps, short for development, security, and operations, is a framework that integrates security practices into every phase of the software development lifecycle. Here is how it works:



1. Continuous Integration and Continuous Deployment (CI/CD)

At the heart of DevSecOps lies the concept of continuous integration and continuous deployment. Developers commit code changes frequently, often multiple times a day. These changes are automatically integrated into a central repository, where automated tests are executed. This early feedback loop ensures that issues are caught early, reducing the risk of vulnerabilities slipping through the cracks.

2. Collaboration and Communication

DevSecOps emphasizes collaboration among cross-functional teams. Developers, security experts, and operations personnel work together from the outset. During planning and design, security considerations are woven into the fabric of the application. This collaborative mindset ensures that security is not an afterthought but an integral part of the development process.

3. Shift Left Security

The term "shift left" refers to moving security practices earlier in the SDLC. Traditionally, security assessments occurred late in the process, often during testing or even after deployment. DevSecOps flips this paradigm. Security testing begins in development environments. Static code analysis, dynamic scans, and vulnerability assessments are performed as part of the build process. By catching vulnerabilities early, organizations reduce the cost and effort required to remediate them.

4. Automated Security Testing

Automation is a cornerstone of DevSecOps. Security tests are automated, ensuring consistent and repeatable results. Common tools include:

- Static Application Security Testing (SAST): Scans source code for vulnerabilities.
- Dynamic Application Security Testing (DAST): Tests running applications for security weaknesses.
- Software Composition Analysis (SCA): Identifies vulnerable third-party libraries.

- Container Security Scans: Checks container images for vulnerabilities.

5. Infrastructure as Code (IaC)

DevSecOps extends beyond application code. Infrastructure as Code (IaC) treats infrastructure components (servers, networks, etc.) as code. Security policies are codified, enabling consistent and secure deployments.

6. Security Culture and Education

DevSecOps isn't just about tools; it's a cultural shift. Organizations foster a security-conscious mindset. Developers receive security training, learning about common vulnerabilities (such as SQL injection, cross-site scripting). Security champions emerge within development teams, advocating for secure practices.

7. Monitoring and Incident Response

DevSecOps extends into production. Continuous monitoring detects anomalies, and incident response plans are rehearsed. Security incidents are treated as learning opportunities, leading to iterative improvements.

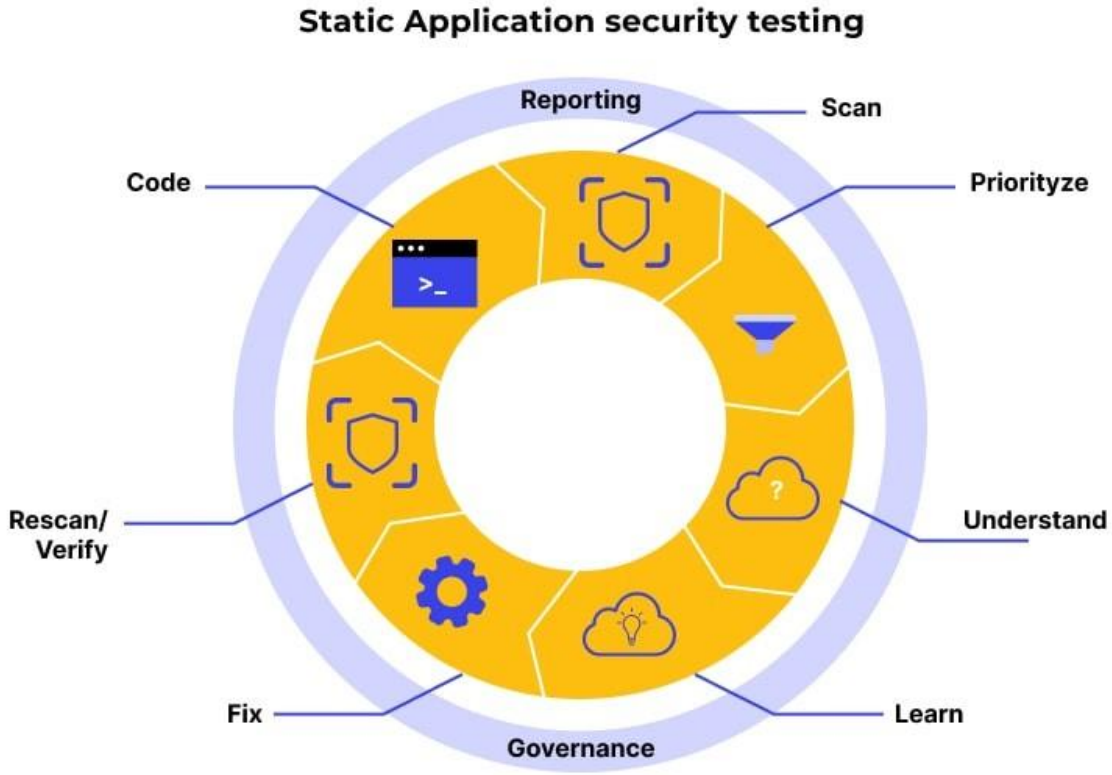
Benefits of DevSecOps

- **Reduced Risk**: By addressing security early, organizations minimize the chances of deploying software with vulnerabilities that attackers could exploit.
- **Faster Time to Market**: DevSecOps streamlines processes, accelerating feature delivery without compromising security.
- **Improved Collaboration**: Silos dissolve, fostering better communication and understanding between teams.
- **Compliance and Auditing**: DevSecOps practices align with compliance requirements, making audits smoother.
- **Resilience**: Organizations respond more effectively to security incidents, minimizing impact.

Static Application Security Testing (SAST)

Source code analysis tools, also known as Static Application Security Testing (SAST) is a type of white-box testing methodology that involves analyzing an application's source code, bytecode, or binary code for vulnerabilities without executing the program. SAST tools scan the code at rest, providing developers with insights into potential security issues that could be exploited by attackers.

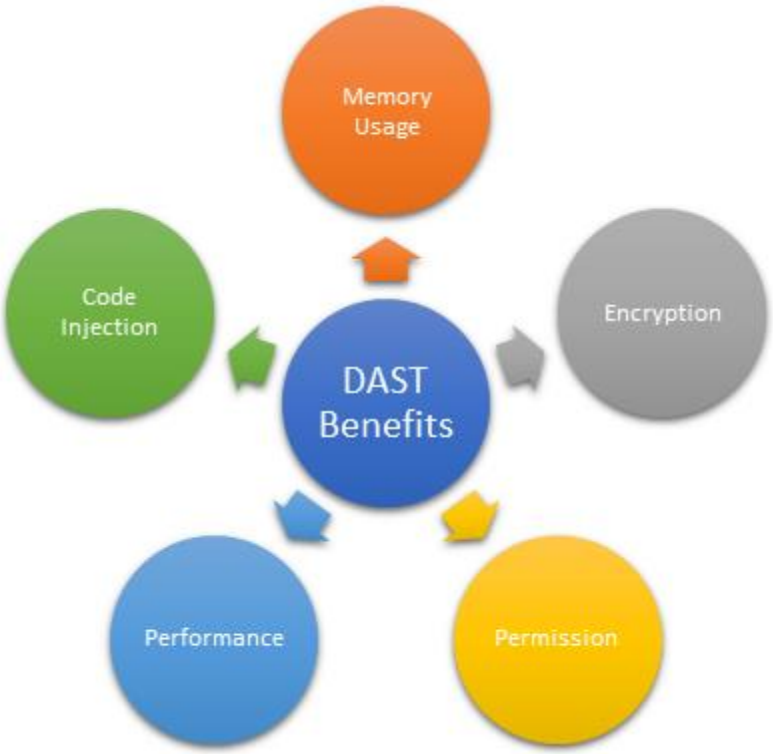
SAST focuses on identifying security flaws, such as coding errors, insecure coding practices, and vulnerabilities like buffer overflows, SQL injection, cross-site scripting (XSS), and others. By examining the code structure, control flow, and data flow, SAST tools help ensure that the application adheres to security best practices before it is deployed.



Dynamic Application Security Testing (DAST)

DAST is a security testing technique designed to identify vulnerabilities in an application by examining its runtime behavior and interactions. This approach is

essential for uncovering security issues that only manifest during the execution of the application and cannot be detected through static analysis alone.



Here are some popular tools:

Checkmarx

Checkmarx provides DAST and SAST via a single platform, ensuring efficient and thorough vulnerability detection. This proactive approach ensures that your software is secure from the beginning. Once vulnerabilities are detected, Checkmarx provides detailed guidance on where and how to fix them, enabling developers to resolve issues quickly and enhance code security.

Key Features:

- Supports over 35 programming languages and more than 80 frameworks, making it a flexible tool for various applications.
- AI integration helps developers by detecting vulnerabilities and offering remediation advice.

- Seamlessly integrates with development environments such as IDEs, version control systems, and CI/CD servers.
- Automated and standardized scan processes enforce security standards consistently.
- Prioritizes vulnerabilities, helping developers understand the risks and severity of each issue flagged.
- Offers guidance on remediation and pinpoints the best location for code fixes.

Use cases: Checkmarx is a robust tool, notable for its effective AI integration, which helps reduce false positives by up to 80%. This capability significantly streamlines the development process, allowing developers to focus on real security issues rather than wasting time on false alerts.

Fortify

Fortify Static Code Analyzer (SCA) is a cybersecurity tool that is designed to identify and address security vulnerabilities within source code. SCA begins by converting source code files into an intermediate format that is optimized for security analysis. This structure is then analyzed to identify security vulnerabilities using an extensive set of secure coding rules and parameters.

Key Features

- Comprehensive database cross-referencing to identify accurately identify a wide range of vulnerabilities and issues
- Ability to identify vulnerabilities across 1,500 categories and types
- Integration with Fortify Software Security Centre (SSC) provides organizations with a centralized management tool and holistic visibility
- Fortify also integrates with multiple IDE's, as well as platforms like Jira, GitHub, Jenkins, and Azure DevOps
- Supports over 27 programing languages, making Fortify suitable for a range of use-cases
- Flexibility allows on-premise, cloud-based, and SaaS deployment

- Audit Assistant uses ML to streamline and enhance vulnerability assessments, thereby reducing time and effort required for manual audits

Use cases: Fortify was selected for this list due to its Depth tuning which can be modified on demand. This allows developers to perform short scans on newly written code, as well as performing in-depth, comprehensive scans on whole projects. The platform's advanced algorithms also ensure that vulnerabilities are efficiently picked up.

Aikido Security

Aikido's Static Application Security Testing (SAST) solution leverages top-tier open-source scanners, including Bandit, Semgrep, and Gosec, along with Aikido's proprietary tools. Aikido's surface monitoring platform dynamically tests for common vulnerabilities in your web app's front end, without reducing performance or breaking any front-end functionality. Teams can customize scanning rules, ensuring the solution is tailored to specific needs. Aikido's risk categorization engine prioritizes finding critical vulnerabilities, such as SQL injection, XSS, and buffer overflows, rather than focusing on non-essential issues like code readability or style.

Key Features:

- IDE Integration: Allows for real-time issue detection as code is being written.
- Broad Language Support: Can be implemented across various use cases due to its extensive language compatibility.
- Detailed Issue Reporting: Provides an analysis of detected issues, including a risk score and remediation suggestions.
- User-Friendly Interface: Features a streamlined UI with easy-to-use reporting capabilities.
- Regulatory Compliance: Ensures adherence to SOC2 and ISO 27001 standards.

Use cases: Aikido is a tool used for both SAST and DAST—it serves as a comprehensive application security platform. Beyond static code analysis, it

addresses Cloud Security Posture Management (CSPM), Software Composition Analysis (SCA), secrets detection, and more. By utilizing multiple scanners, Aikido effectively identifies a wide array of security concerns, including cloud misconfigurations, vulnerable dependencies, and malware.

Cycode SAST

Cycode's SAST solution is designed with a focus on speed, precision, and user-friendliness. It offers a comprehensive Application Security Posture Management (ASPM) approach, featuring proprietary code-scanning capabilities that span from code development to cloud environments. This modern SAST solution integrates seamlessly with over 100 pre-built connections to third-party security tools, providing real-time visibility throughout the Software Development Life Cycle (SDLC). Cycode's SAST ensures swift and accurate code analysis, enabling developers to quickly address and resolve issues.

Key Features:

- Supports a wide range of programming languages and frameworks, including Java, PHP, C#, Python, Swift, and C.
- AI-driven SAST engine that delivers intelligent, context-aware remediation suggestions.
- Easily integrates with developer environments such as IDEs and CLI, with rapid scanning of Pull Requests.
- Provides fast, continuous real-time scanning capabilities.
- Prioritizes vulnerabilities based on business impact and risk score.
- Protects against sensitive data exfiltration risks.
- Offers a complete ASPM solution, combining SAST scanning with broader security visibility across the organization.

Use cases: Cycode's SAST solution provides a holistic ASPM platform that secures the entire software supply chain from development to deployment in the cloud. In addition to SAST, it offers solutions for secrets management, software composition analysis, CI/CD security, infrastructure as code (IAC) scanning, and container security. The Risk Intelligence Graph (RIG) further enhances security by

tracking code integrity and events across the SDLC, helping teams prioritize risks and detect anomalies throughout the organization's ecosystem.

Here's a detailed comparison between Static Application Security Testing (SAST) and Dynamic Application Security Testing (DAST) presented in table format:

S. No.	Static Application Security Testing (SAST)	Dynamic Application Security Testing (DAST)
1	SAST is a type of White Box security testing.	DAST is a type of Black Box security testing.
2	In SAST, the application is tested from inside out.	In DAST, the application is tested from outside in.
3	This type of testing is a developer's approach.	This type of testing is a hacker's approach.
4	No deployed application is required for SAST.	A running application is required for DAST.
5	Finding vulnerabilities, identifying, and fixing bugs is easier in SAST.	Finding vulnerabilities towards the end of the SDLC.
6	Fixing vulnerabilities is possible with little cost assistance.	It finds vulnerabilities towards the end of the SDLC, hence it is expensive to fix.
7	SAST cannot discover issues related to runtime and environment.	DAST can discover issues related to runtime and environment.
8	Typically supports all types of software like web applications, web services, thick clients.	Typically only scans apps like web applications and web services, not other types of software.

9	In SAST, the developer has knowledge about design, application framework, and implementation.	In DAST, the tester has no knowledge about the application, design, frameworks, and implementation.
10	SAST testing requires source code to perform testing operations.	DAST testing does not require source code to perform testing operations.
11	As it scans static code and performs its testing operation, it is called Static Application Security Testing (SAST).	As it scans dynamic code and performs its testing operation, it is called Dynamic Application Security Testing (DAST).
12	This testing is performed in the early stages of the Software Development Life Cycle (SDLC).	This testing is performed at the end of the Software Development Life Cycle (SDLC).

Securing Web Applications

To safeguard web applications effectively, follow these essential practices:

1. Implement Secure Coding Practices

Ensure security from the coding phase by:

- **Validating Inputs:** Check all user inputs to block harmful data.
- **Encoding Outputs:** Encode data before displaying it to prevent XSS attacks.
- **Using Parameterized Queries:** Protect against SQL injection with prepared statements.
- **Handling Errors Properly:** Avoid disclosing sensitive information through error messages.

2. Strengthen Authentication and Authorization

Enhance access control by:

- **Using MFA:** Add an extra security layer with multi-factor authentication.
- **Enforcing Strong Passwords:** Require complex passwords and use secure hashing methods.
- **Managing Sessions Securely:** Implement secure session handling practices.

3. Secure Data Transmission

Protect data by:

- **Enforcing HTTPS:** Encrypt data transmission to prevent interception.
- **Encrypting Data:** Secure sensitive data both in transit and at rest.

4. Conduct Regular Security Testing

Identify vulnerabilities by:

- **Performing SAST and DAST:** Use static and dynamic testing to find and fix issues.
- **Engaging in Penetration Testing:** Simulate attacks to uncover hidden vulnerabilities.

5. Monitor and Respond to Security Threats

Stay vigilant by:

- **Implementing Logging and Monitoring:** Track and analyze activities to detect security issues.
- **Preparing an Incident Response Plan:** Have a strategy for addressing security breaches.

6. Maintain Secure Configurations and Updates

Ensure ongoing security by:

- **Applying Secure Configurations:** Set up systems securely and disable unnecessary features.
- **Keeping Software Updated:** Regularly update software and dependencies to patch vulnerabilities.

7. Educate and Train Your Team

Build awareness by:

- **Providing Security Training:** Educate staff on best practices and emerging threats.
- **Conducting Code Reviews:** Regularly review code to catch potential security issues.

8. Follow Compliance and Standards

Adhere to industry guidelines by:

- **Following Security Standards:** Implement best practices from frameworks like OWASP.
- **Ensuring Regulatory Compliance:** Meet legal requirements for data protection.

9. Manage Third-Party Components

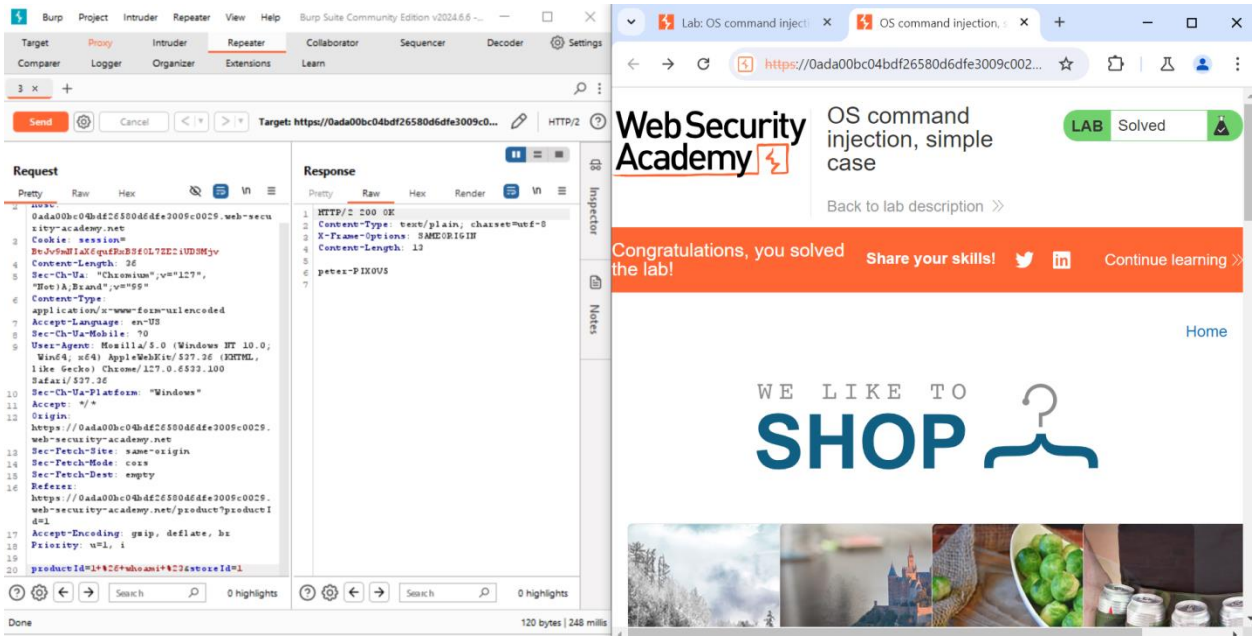
Safeguard your application by:

- **Assessing Dependencies:** Regularly check third-party components for vulnerabilities.
- **Ensuring Vendor Security:** Verify that third-party vendors adhere to security standards.

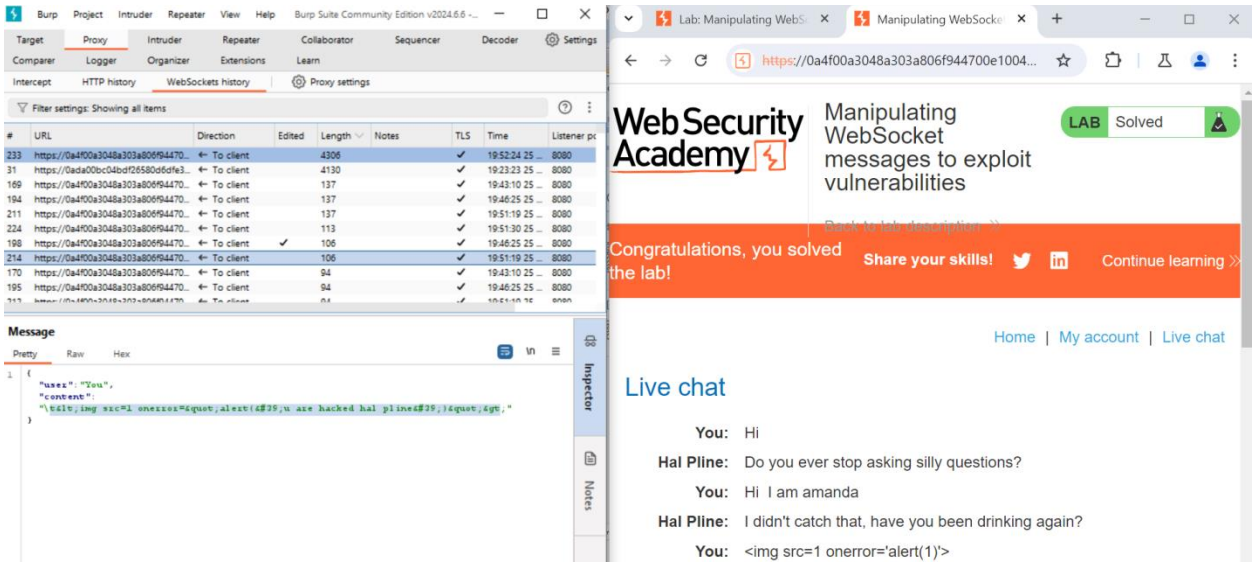
Practical:

Here are the screenshots of completion of labs

1. https://portswigger.net/web-security/os-command-injection/lab-simple



2. https://portswigger.net/web-security/websockets/lab-manipulating-messages-to-exploit-vulnerabilities



References:

- https://www.microsoft.com/en-us/security/business/security-101/what-is-devsecops
- https://www.ibm.com/topics/devsecops
- https://medium.com/@anjunairateam/step-by-step-guide-to-conducting-web-application-security-assessment-02d125efc7fc
- https://owasp.org/www-community/Source_Code_Analysis_Tools
- https://expertinsights.com/insights/the-top-static-application-security-testing-sast-tools/
- https://expertinsights.com/insights/the-top-dynamic-application-security-testing-dast-tools/
- https://www.synopsys.com/blogs/software-security/sast-vs-dast-difference.html
- https://www.geeksforgeeks.org/securing-web-applications/
- https://www.breachlock.com/resources/blog/benefits-of-dast-testing-for-application-security/