0Application Security Verification Standard (ASVS):

The **Application Security Verification Standard (ASVS)** is a checklist or set of guidelines created by the **OWASP Foundation** to help developers, testers, and security professionals build and verify secure web and mobile applications.

Think of ASVS as a **security rulebook** for applications. It provides a list of security requirements that an app should meet to be considered safe from hackers. It covers everything from authentication (login security) to data protection and error handling.

It helps developers, testers, and security teams ensure their applications are protected against common attacks like:

1. Cross-Site Scripting (XSS)
2. SQL Injection
3. Broken Authentication
4. Data Leaks

ASVS Structured Levels

The OWASP Application Security Verification Standard (ASVS) categorizes security requirements into three levels based on risk and application criticality.

Level 1 (Basic Security) – "Essential Protection"

It is for Low-risk applications (e.g., blogs, simple websites). Its goal is to prevent **common attacks** and meet **minimum security standards**. Blocks common attacks like SQLi/XSS. Requires HTTPS, password policies, and basic input validation. Minimal security for non-sensitive data.

Level 2 (Standard Security) – "Strong Protection"

It is for Sensitive applications (e.g., e-commerce, healthcare, banking). The main Goal is to defend against targeted attacks and comply with industry regulations (PCI DSS, HIPAA). Adds MFA (Multi Factor Authentication), encryption at rest, secure APIs, and Apply role-based access control (RBAC). Re-authentication for critical operations (e.g., changing passwords).

Level 3 (Advanced Security) – "Military-Grade Protection"

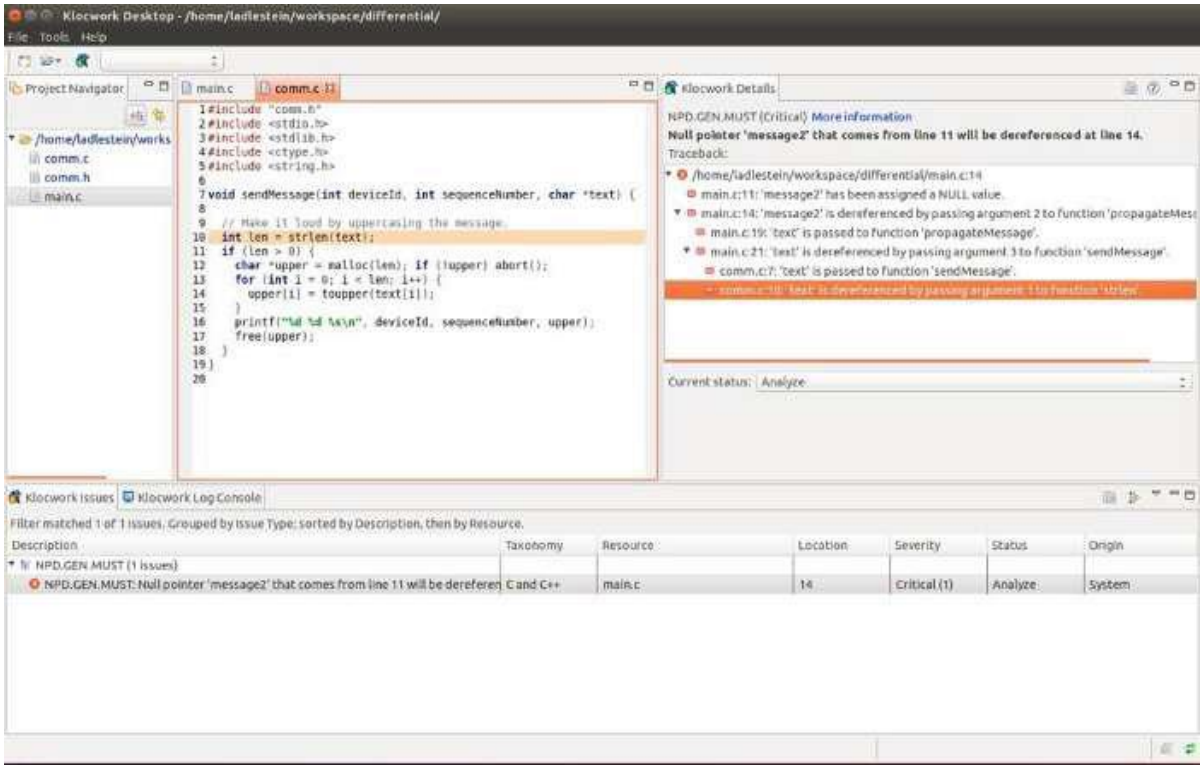
It is for Mission-critical systems (e.g., defense, financial transactions, government). The goal is to defend against advanced persistent threats (APTs) and nation-state attackers. Hardened Authentication like Hardware-based MFA (e.g., smart cards), Continuous authentication checks (e.g., behavioral biometrics). Zero-Trust Architecture, like Micro-segmentation, least privilege access, and Runtime application self-protection (RASP)

SAST and DAST:

SAST (Static Application Security Testing) and **DAST (Dynamic Application Security Testing)** are two fundamental methods in application security testing, each serving unique roles within the software development lifecycle.

Static Application Security Testing (SAST) examines the application's source code, bytecode, or binaries for vulnerabilities without executing the program. It is a *white-box testing* method, meaning the tester has access to the inside of the application—its code, design, and frameworks. Typically used early in the Software Development Lifecycle (SDLC), SAST helps developers identify issues such as SQL injection, buffer overflows, and code quality problems before deployment.

For example, for Klockwork’ s users, once you have established a link between your project and Klockwork Desktop, it allows you to code your program normally using any IDE of your choice as long as it is open in the background. Each time you save the file, Klockwork Destop will update the code automatically and perform a scan on the spot. If it detects any security issues, it will display them on the user interface.



Strengths: Allows early detection of vulnerabilities, provides comprehensive code coverage, and does not require a running application.

Limitations: May produce a high number of false positives, cannot detect runtime or environment-specific issues, and can struggle with complex environments or dynamic code paths.

Dynamic Application Security Testing (DAST) scans a running application for vulnerabilities by simulating external attack scenarios, much like an attacker would.
It is a *black-box testing* method, meaning the tester does not need access to the source code—only the running application.
Performed later in the SDLC, usually just before or after deployment, DAST finds issues such as cross-site scripting (XSS), broken authentication, and other vulnerabilities that appear only at runtime.
Strengths: Can detect runtime and environmental vulnerabilities, does not require access to source code, and provides real-world attack simulation.
Limitations: May miss code-level issues, is limited to what's accessible from the outside, and fixing detected vulnerabilities late in the SDLC can be more expensive.

Comparison Table

SAST	DAST
White-box testing (inside-out)	Black-box testing (outside-in)
Needs source code or binaries	Needs a running, deployed application
Detects vulnerabilities in code early	Detects issues at runtime or deployment
Fast feedback for developers	Simulates real-world attacks
Cannot find runtime/config issues	Cannot find all code-level vulnerabilities
Used early in SDLC	Used later in SDLC

What is Interactive Application Security Testing (IAST)?

Interactive Application Security Testing (IAST) is a modern security testing approach that analyzes applications for vulnerabilities while they are running, by embedding agents or sensors directly inside the application's runtime environment. Unlike SAST (which analyzes static code) and DAST (which tests from outside while the app runs), IAST operates within the application as it is exercised by manual or automated tests.

Real-time vulnerability detection: IAST works while the application is running, identifying vulnerabilities as they are triggered during functional testing or user interactions.
Contextual analysis: It provides detailed context, such as the exact line of code and the HTTP request/response that triggered the issue, enabling developers to reproduce and fix problems quickly.
Comprehensive coverage: IAST examines both code and runtime environment, allowing for detection of vulnerabilities that may only appear during execution (e.g., those resulting from specific data paths, configurations, or code interactions).
Ease of integration: The agent-based approach enables seamless integration into CI/CD pipelines, making IAST a preferred choice for DevSecOps and agile development environments.

Applications:
Web application security: Identify SQL injection, XSS, insecure configurations, and more during QA or in staging.
DevSecOps: Continuous security testing integrated with automated builds and deployments.
Regulatory compliance: Meet requirements for secure coding practices by identifying and addressing vulnerabilities early in the SDLC.

Limitations:
Requires running tests: IAST needs the application to be exercised, either through manual use, automated functional testing, or API requests.
Complex environments: Instrumentation complexity can be higher in certain application architectures or technology stacks.

What is OWASP ZAP?
OWASP ZAP (Zed Attack Proxy) is an open-source Dynamic Application Security Testing (DAST) tool designed to find security vulnerabilities in web applications while they are running. Developed by an international team of volunteers and maintained under the OWASP (Open Web Application Security Project) umbrella, ZAP is one of the most widely used free application security tools for developers, testers, and security professionals.

Open source: ZAP is free to use and benefits from continuous innovation and updates from a large global community.
Dynamic scanning: As a DAST tool, it simulates real-world attacks by actively interacting with running applications to discover vulnerabilities (such as SQL injection, XSS, and others).
Automated and manual testing: ZAP supports both automated scanning and manual penetration testing, including capabilities for crawling, spidering, and intercepting traffic.
Authentication support: ZAP can scan parts of an application that require user login, making assessments more comprehensive.
Easy integration: ZAP can be integrated into CI/CD pipelines and used via a desktop application or through its REST API for automation.
Vulnerability coverage: It detects a range of issues, including missing security headers, exposure of sensitive information, cross-site scripting, file inclusion, and more.

Applications:
Development security: ZAP is regularly used in SDLC pipelines to catch vulnerabilities before release.
Penetration testing: Popular with security professionals who want both automated and granular manual analysis options.
Compliance: Helps meet security best practices and standards by testing for OWASP Top 10 vulnerabilities.

What Is the Difference Between Active & Passive Scan?

Feature	Passive Scan	Active Scan
Interactivity	Observes only (no changes to requests)	Actively modifies/sends attack requests
Safety	Very safe, no impact on app	Potentially disruptive, not recommended on prod
Coverage	Limited (surface-level weaknesses)	Broad (including deeper vulnerabilities)
Typical Usage	Continuous monitoring, initial testing	Pre-production, controlled environments, pen-testing
Example Issues	Missing headers, info leak	XSS, SQLi, command injection

What is Manual Penetration Testing?

Manual penetration testing is the testing that is done by human beings. In such type of testing, the vulnerability and risk of a machine is tested by an expert engineer.

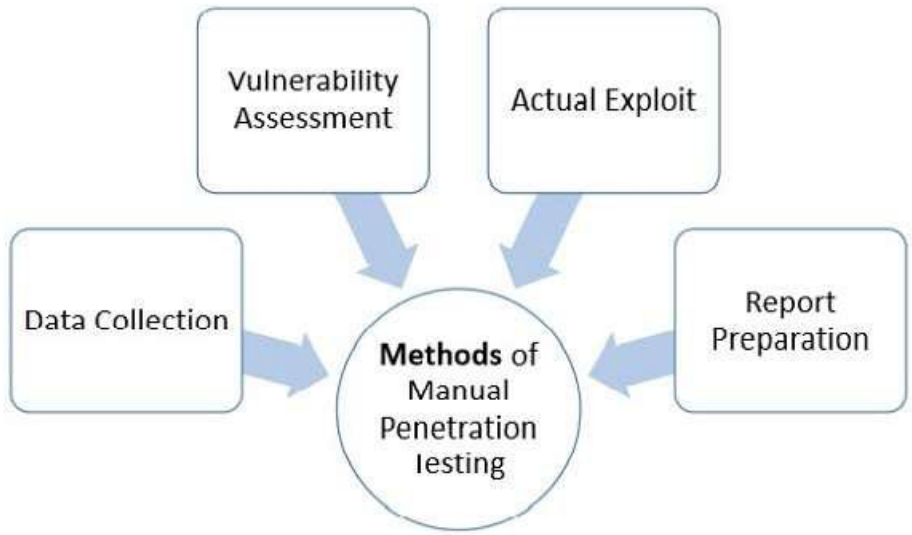
Generally, testing engineers perform the following methods −

Data Collection − Data collection plays a key role in testing. One can either collect data manually or can use tool services (such as webpage source code analysis technique, etc.) freely available online. These tools help to collect information like table names, DB versions, database, software, hardware, or even about different third-party plugins, etc

Vulnerability Assessment − Once the data is collected, it helps the testers to identify the security weakness and take preventive steps accordingly.

Actual Exploit − This is a typical method that an expert tester uses to launch an attack on a target system, and likewise, reduces the risk of attack.

Report Preparation − Once the penetration is done, the tester prepares a final report that describes everything about the system. Finally, the report is analyzed to take corrective steps to protect the target system.



Types of Manual Penetration Testing

Manual penetration testing is normally categorized in two ways −

Focused Manual Penetration Testing – It is a much focused method that tests specific vulnerabilities and risks. Automated penetration testing cannot perform this testing; it is done only by human experts who examine
specific application vulnerabilities within the given domains.
Comprehensive Manual Penetration Testing – This involves testing the entire system, including all interconnected components, to identify various risks and vulnerabilities. However, the function of this testing is more situational, such as investigating whether multiple lower-risk faults can bring a more vulnerable attack scenario, etc

Runtime Application Self-Protection (RASP):

Runtime Application Self-Protection (RASP) is a security technology that protects applications from real-time attacks while they are running. Unlike perimeter-based solutions like firewalls, RASP works from inside the application, monitoring inputs, outputs, and internal behavior to detect and stop threats as they occur.

How RASP Works

Integration: RASP is embedded within the running application, either as a framework, module, or library, and does not require architectural code changes. Sensors and controls activate as soon as the application starts.
Continuous Monitoring: It inspects all incoming requests, user behavior, data flows, and application execution paths, looking for suspicious or malicious activity in the context of the current state of the application.
Detection: If a security threat or vulnerability—such as code injection, data theft, or zero-day exploit—is detected, RASP responds in real time by alerting, blocking, or mitigating the attack.
Real-time Protection: RASP can take actions such as blocking malicious requests, virtually patching the application, terminating user sessions, or shutting the application down to prevent further damage.
Minimal False Positives: Since RASP operates with detailed contextual knowledge of the application logic, data, and configuration, it can more accurately distinguish between normal operations and attacks, reducing false positives.
Example Scenario
If an attacker tries a SQL injection, RASP detects and prevents the execution of suspicious database instructions—not just by recognizing the signature of the attack, but by observing abnormal behavior within the application as it happens.
Benefits

- Detects and blocks attacks in real time
- Protects against known and unknown (zero-day) vulnerabilities
- Delivers minimal false positives due to contextual analysis
- Requires minimal changes to existing application code or architecture

RASP bridges the gap left by traditional testing and network controls by offering deep visibility into runtime behavior and providing immediate, application-centric protection.

Key Benefits:

Contextual Awareness: When a RASP solution identifies a potential threat, it has additional contextual information about the current state of the application and what data and code is

affected. This context can be invaluable for investigating, triaging, and remediating potential vulnerabilities since it indicates where the vulnerability is located in the code and exactly how it can be exploited.

Visibility into Application-Layer Attacks: RASP has deep visibility into the application layer because it is integrated with a particular application. This application-layer visibility, insight, and knowledge can help to detect a wider range of potential attacks and vulnerabilities.

Zero-Day Protection: While RASP can use signatures to identify attacks, it is not limited to signature-based detection. By identifying and responding to anomalous behaviors within the protected application, RASP can detect and block even zero-day attacks.

Lower False Positives: RASP has deep insight into an application's internals, including the ability to see how a potential attack affects the application's execution. This dramatically increases RASP's ability to differentiate true attacks (which have a true negative impact on application performance and security) from false positives (such as SQL injection attempts that are never included in an SQL query). This reduction in false positives decreases load on security teams and enables them to focus on true threats.

Lower CapEx and OpEx: RASP is designed to be easy to deploy, yet is able to make a significant difference in an application's vulnerability to attack and rate of false positive alerts. This combination reduces both upfront expenses (CapEx) and the cost of effectively protecting the application (OpEx) compared to manual patching and web application firewalls (WAFs).

Easy Maintenance: RASP works based upon insight into an application, not traffic rules, learning, or blacklists. SOC teams love this reliability and CISOs appreciate the resource savings. Applications become self-protected and remain protected wherever they go.

Flexible Deployment: While RASP is typically based upon HTML standards, it is easy to adapt its API to work with different standards and application architectures. This enables it to protect even non-web applications using standards like XML and RPC.

Cloud Support: RASP is designed to integrate with and be deployed as part of the application that it protects. This enables it to be deployed in any location where the protected applications can run, including in the cloud.

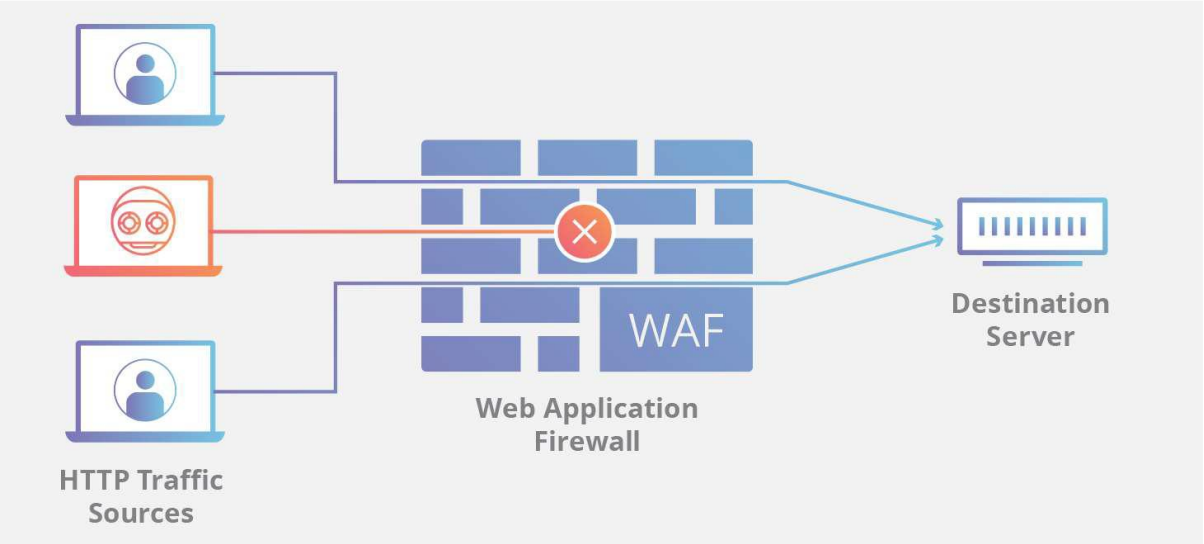
Define Web Application Firewall. Demonstrate using a tool.

A Web Application Firewall (WAF) is a security solution that filters, monitors, and blocks HTTP/HTTPS traffic to and from a web application. Its primary role is to protect web applications by acting as a shield between the web app and the internet, filtering out malicious traffic and preventing attacks such as SQL injection, cross-site scripting (XSS), cross-site request forgery, file inclusion, and other common web vulnerabilities. WAFs operate mostly at the application layer (Layer 7 of the OSI model) and can be deployed as hardware appliances, software, or cloud-based services. They use a set of rules or policies to identify and stop potentially harmful traffic before it reaches the application, providing real-time defense against web threats and helping to mitigate zero-day vulnerabilities and DDoS attacks.

Demonstration of a Web Application Firewall Tool

One widely known example is AWS Web Application Firewall (AWS WAF), which helps protect publicly facing web endpoints from malicious actors by inspecting HTTP(S) requests and applying custom rules.

A WAF operates through a set of rules often called policies. These policies aim to protect against vulnerabilities in the application by filtering out malicious traffic. The value of a WAF comes in part from the speed and ease with which policy modification can be implemented, allowing for faster response to varying attack vectors; during a DDoS attack, rate limiting can be quickly implemented by modifying WAF policies.



Types of Web Application Firewalls

There are three primary ways to implement a WAF:

Network-based WAF−usually hardware-based, it is installed locally to minimize latency. However, this is the most expensive type of WAF and necessitates storing and maintaining physical equipment.

Host-based WAF−can be fully integrated into the software of an application. This option is cheaper than network-based WAFs and is more customizable, but it consumes extensive local server resources, is complex to implement, and can be expensive to maintain. The machine used to run a host-based WAF often needs to be hardened and customized, which can take time and be costly.

Cloud-based WAF−an affordable, easily implemented solution, which typically does not require an upfront investment, with users paying a monthly or annual security-as-a-service subscription. A cloud-based WAF can be regularly updated at no extra cost, and without any effort on the part of the user. However, since you rely on a third party to manage your WAF, it is important to ensure cloud-based WAFs have sufficient customization options to match your organization’s business rules.

What are Standard Operating Procedures (SOP)?

A Standard Operating Procedure (SOP) is a **set of step-by-step written instructions** designed to help workers carry out routine operations to ensure consistency, quality, and efficiency. SOPs reduce miscommunication, ensure compliance with regulations, and promote uniformity in performing tasks across an organization. They serve as a documented framework for how processes or activities should be performed to maintain quality standards and operational control.

Types of Standard Operating Procedures (SOP)

SOPs can be classified into several types based on their purpose and application:

1. **Production SOPs:** Instructions on operating machines or equipment.
2. **Assembly Procedures:** Guidelines for assembling products or systems.
3. **Inspection Procedures:** Quality control and inspection guidelines.
4. **Security SOPs:** Occupational safety and risk-minimizing instructions.
5. **Emergency Procedures:** Steps for handling emergencies.
6. **Maintenance SOPs:** Preventive and corrective maintenance.
7. **Quality SOPs:** Ensure products/services meet quality standards.
8. **Administrative SOPs:** Document management and internal workflow.
9. **Customer Service Procedures:** Guidelines for handling customer issues and service quality.

Why Do You Need Standard Operating Procedures (SOP)?

1. **Boost Efficiency and Reduce Costs:** SOPs streamline tasks, eliminate guesswork, and minimize errors, leading to significant time and cost savings.
2. **Ensure Consistency and Quality Control:** Uniform procedures enhance performance consistency and maintain quality across operations.
3. **Support Compliance and Risk Management:** Essential for regulated industries to meet legal and safety standards.
4. **Reduce Employee Turnover:** Clear instructions empower employees, reduce frustration, and improve retention.
5. **Enable Scalability and Adaptability:** SOPs provide a scalable process framework that can evolve with business growth.
6. **Preserve Knowledge:** SOPs minimize knowledge loss when key employees are absent or leave.

Steps for Writing a Standard Operating Procedure (SOP):

1. **Define the Scope:** Clearly determine the purpose, boundaries, and specific problem the SOP addresses.

2. **Gather Information:** Collect detailed insights about the current process, involving subject matter experts when necessary.
3. **Choose the Format:** Decide on the most suitable SOP format—text document, checklist, flowchart, or visual aids.
4. **Complete the Draft:** Write the first draft, organizing steps consecutively.
5. **Document Step-by-Step Instructions:** Use simple, clear, action-oriented language from the user's perspective.
6. **Include Visual Aids:** Add diagrams, flowcharts, or images for better comprehension.
7. **Consider Safety and Quality Measures:** Explicitly outline any relevant safeguards or standards.
8. **Review and Validation:** Engage stakeholders or experts to verify the SOP's accuracy and practicality.
9. **Implement and Train:** Use the SOP to train employees and ensure proper adoption.
10. **Regular Review and Updates:** Periodically revise and improve the SOP to keep it current.
11. **Document Control:** Maintain versioning and ensure access to the latest SOP documents.
12. **Measure Effectiveness:** Continuously monitor and optimize SOP performance through feedback and data.

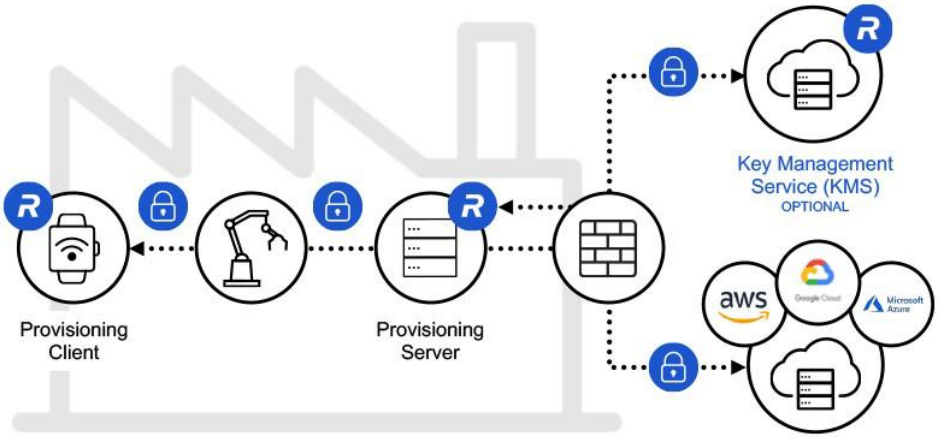
SOP Best Practices:

1. **Identify Your Audience and Goals:** Tailor the SOP to the target users and purpose.
2. **Involve Your Team:** Collaborate with relevant personnel to gather comprehensive details.
3. **Write Concisely and Clearly:** Avoid jargon, use plain language, and active voice.
4. **Avoid Ambiguity:** Use precise instructions with unambiguous action verbs.
5. **Use Formatting:** Bullet points and lists improve clarity and readability.
6. **Distribute Digitally:** Use digital platforms for easy access, updates, and tracking.
7. **Continuously Review:** Regularly refresh SOPs and incorporate feedback.
8. **Leverage SOP Software:** Tools like Scribe can assist in creating, managing, and updating SOPs efficiently.

Secure Device Provisioning:

Secure Device Provisioning is the process of **configuring and initializing a device with the necessary software, settings, and security credentials** to ensure its secure operation within a network or system. This process is vital for protecting devices from cyber threats by securely loading firmware, encryption keys, certificates, and network configurations during manufacturing or deployment. It guarantees that device secrets are not exposed or tampered with and maintains protection throughout the device’s lifecycle even without requiring specialized hardware security resources on the device.

How Secure Device Provisioning Works



1. **Device Initialization:** The device is prepared with the required firmware and software to get it started.
2. **Security Credential Loading:** The device is securely injected with important secrets such as encryption keys, cryptographic certificates, and identity data to enable trusted communication.
3. **Configuration:** Device-specific settings, including network parameters and security policies, are applied to ensure the device operates in a compliant and secure manner.
4. **Verification:** The provisioned device undergoes checks to confirm that it has been correctly configured and that its security credentials are intact and valid.
5. **Continuous Protection:** Technologies such as secure boot, whitebox cryptography, and code protection help maintain the integrity and secrecy of the device's credentials throughout its operational life.
6. **Remote Monitoring and Control:** In some implementations, device manufacturers or administrators can remotely oversee and control the provisioning process without physical presence at the manufacturing site.

Secure device provisioning involves secure communication protocols to exchange sensitive data and often includes network access control mechanisms that evaluate device compliance before granting network connectivity. When integrated with Network Access Control (NAC), provisioning can enforce authentication, configuration, and remediation to keep devices secure on the network.

Deployment:

An automation process workflow must be deployed to send it to the server. From server, the deployed process is assigned to a robot for execution.

The Deployment tab allows you to deploy the published processes in Automation Studio. It also enables you to decommission the current version, restore the previous version and add a process as a process bot. Published as well as deployed processes are available in this tab. You can view details related to the process, such as the name of the process, its type, assigned profile, version, and so on.

The version of the process displayed in the Deployment tab is the version that gets created while publishing the process. If you edit a deployed process, save and publish the deployed process again. The version of the saved and published process post-deployment gets incremental by 1. Every version of the saved process that you publish is available for deployment.

OWASP Software Assurance Maturity Model (SAMM) :

The OWASP Software Assurance Maturity Model (SAMM) is an **open framework designed to help organizations formulate and implement a strategy for software security** that is customized to the specific risks and needs of the organization. It provides an effective, measurable, and structured way to evaluate an organization's existing software security practices, build a balanced software security assurance program in iterations, and demonstrate tangible improvements over time.

Key Characteristics of OWASP SAMM:

1. **Covers the complete software lifecycle**: from development and acquisition to deployment and operations.
2. **Technology and process agnostic**: Can be applied regardless of the technology stack or development methodology (Agile, Waterfall, DevOps).
3. **Evolutive and risk-driven**: The model allows organizations to evolve their security processes based on risk prioritization.
4. **Prescriptive and measurable**: It defines clear objectives, activities, and measurable outcomes across different maturity levels.

Structure:

OWASP SAMM is organized around **five key business functions** that represent critical areas of software assurance:

1. **Governance** - Strategy, metrics, policy, and compliance.
2. **Design** - Threat assessment, secure architecture, and security requirements.
3. **Implementation** - Secure coding, code review, and security testing.
4. **Verification** - Testing and review practices to identify vulnerabilities.
5. **Operations** - Environment hardening, vulnerability management, and operational enablement.

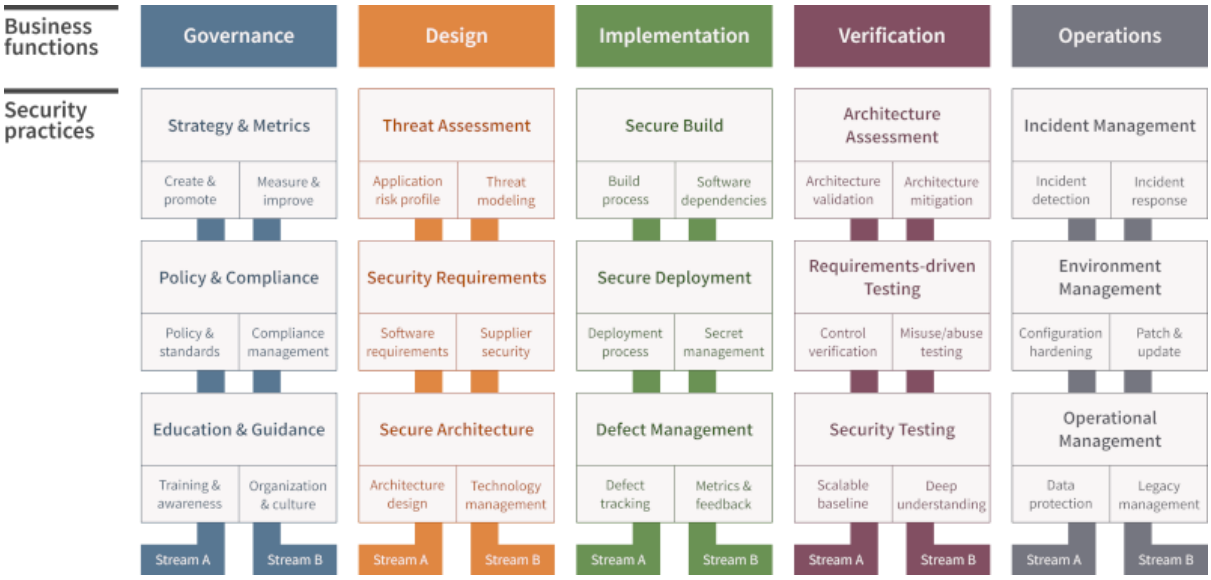
Each function consists of **security practices** divided into streams, with defined maturity levels (0 to 3) that indicate the sophistication and effectiveness of that practice in the organization.

Benefits:

1. Aligns security activities with business risks and objectives.
2. Provides a roadmap for incremental improvement in software security.
3. Offers a common language for security and management teams.
4. Helps demonstrate progress and ROI for security investments.

Usage:

1. Organizations can perform self-assessments or involve third parties to evaluate their maturity level. They can then use SAMM to plan, prioritize, and implement security improvements in a systematic and measurable manner.
2. In summary, OWASP SAMM is a comprehensive and scalable model that enables organizations to systematically improve their software security posture throughout the entire software lifecycle, making it an essential framework for attaining software assurance maturity.



Definition of Metrics

Metrics are **quantifiable measurements** used to assess performance, track progress, and evaluate the success of various processes, initiatives, or entities. They provide objective data that helps organizations make informed decisions, identify areas for improvement, and monitor strategy effectiveness. Metrics serve as benchmarks and indicators, enabling comparisons over time or against established goals.

Types of Metrics

Here are common categories of metrics used in organizations:

1. **Operations Metrics:** Measure the performance and effectiveness of operational processes, such as uptime, throughput, or cycle time.
2. **Efficiency Metrics:** Assess how resources are used relative to outputs, including productivity rates and cost per unit.
3. **Quality Metrics:** Evaluate the quality of products, services, or processes. Examples include defect rates, customer satisfaction score, and first-time pass rates.
4. **Financial Metrics:** Track monetary performance such as revenue, profit margins, or return on investment.
5. **Customer Metrics:** Gauge customer experience and loyalty, including Net Promoter Score (NPS) and Customer Effort Score (CES).
6. **Safety Metrics:** Measure compliance with safety protocols and incident rates.
7. **Employee Performance Metrics:** Evaluate workforce productivity and engagement.
8. **Environmental Metrics:** Measure sustainability and ecological impact.

Example: Application Security Metrics from OWASP

OWASP provides specific metrics to evaluate the effectiveness of application security programs:

1. **Vulnerability Discovery Time:** Time elapsed from code deployment to identification of a vulnerability. Useful to measure the effectiveness of security testing.
2. **Time to Remediate:** Duration taken to fix a vulnerability from its discovery. Indicates responsiveness of the security and development teams.
3. **Patch Deployment Time:** Time from patch release to deployment in the live environment.

4. **Vulnerability Density:** Number of vulnerabilities found per size of code (e.g., per 1000 lines).
5. **Risk Coverage:** Percentage of identified security risks that have been assessed and mitigated.
6. **Vulnerability Severity Levels:** Categorization of vulnerabilities based on severity (Critical, High, Medium, Low). Helps prioritize remediation efforts.