The Building Security In Maturity Model (BSIMM)

A Practical Implementation Guide & Reference

Version: 1.0
Author: Ivan Piskunov
Date: 2024/2025

Document Preface

This document is a concise, action-oriented reference guide for implementing the Building Security In Maturity Model (BSIMM) practices. It distills the key activities from the framework into actionable workflows, control implementation guides, and process creation checklists.

Purpose: To serve as a practical handbook for engineers, security champions, and team leads to build and mature a Software Security Initiative (SSI) based on real-world data from the BSIMM community.

Intended Audience:

AppSec Engineers: For building and scaling security programs.

DevSecOps Teams: For integrating security controls into CI/CD pipelines.

Security Champions: For understanding and promoting security within their teams.

Development Team Leads: For implementing security processes within development lifecycles.

CISOs & BISOs: For strategizing and measuring the progress of their SSI.

Table of Contents

1. BSIMM in a Nutshell

1.1 What is BSIMM? Core Philosophy

BSIMM is a **descriptive framework** based on data gathered from over 120 companies. It doesn't tell you *what to do* but shows you what other successful organizations **are actually doing**. It consists of **4 Domains**, **12 Practices**, and **128 observable Activities** that you can implement based on your maturity level.

1.2 How to Use This Guide

This guide translates BSIMM activities into actionable steps. For each practice, you will find:

Implementation Workflow: A step-by-step process to build the practice.

Control Checklist: A list of specific controls to implement.

Integration Tips: How to plug it into your existing SDLC and tools.

2. Domain 1: Governance

Focus: Creating the foundation, rules, and strategy for your SSI.

2.1 Practice: Strategy & Metrics

Goal: Define and measure your software security program.

Implementation Workflow:

Define SSI Goals: Align with business objectives (e.g., "Reduce critical vulnerabilities in new features by 50% in 6 months").

Assign Roles: Appoint an **AppSec Lead**, identify **Security Champions** in each dev team.

Establish Metrics: Track key metrics from day one:

```
# of security assessments completed
% of projects with threat models
Mean Time To Remediate (MTTR) vulnerabilities
Security test coverage (%)
```

2.2 Practice: Compliance & Policy

Goal: Establish security requirements and manage third-party risk.

Control Checklist:

- [ ] **Create a Security Policy:** A short, clear document mandating security activities (e.g., "All new services must undergo a threat modeling session").
- [ ] **Define Security Requirements:** Incorporate OWASP ASVS or CWE top 25 into your definition of "Done."
- [ ] **Third-Party Risk Management:**

Require SBOM (Software Bill of Materials) for all acquired software.

Integrate SCA (Software Composition Analysis) tools like Snyk, Mend (formerly WhiteSource) into your CI to automatically block builds with critical library vulnerabilities.

2.3 Practice: Training

Goal: Provide role-specific security knowledge.

Security Champions Program Setup:

Identify Volunteers: Find 1-2 interested developers per team.

Provide Training: Equip them with basic AppSec knowledge (OWASP Top 10, secure coding).

Empower Them: Give them authority to request security reviews for their team's projects.

Create a Feedback Loop: Regular meetings for champions to share issues and best practices.

3. Domain 2: Intelligence

Focus: Building and sharing security knowledge across the organization.

3.1 Practice: Attack Models

Goal: Understand threats to your software.

Threat Modeling Process (Simple & Agile):

Diagram: Create a simple data flow diagram (DFD) of the application.

Identify Threats: Use the STRIDE model (Spoofing, Tampering, Repudiation, Information Disclosure, DoS, Elevation of Privilege).

Mitigate: Decide on mitigations for each threat (e.g., "Threat: Tampering with data in transit -> Mitigation: Enforce TLS 1.3").

Tool: Use OWASP Threat Dragon or draw.io. Integrate this as a mandatory step in the design phase for all new features.

3.2 Practice: Standards & Requirements

Goal: Provide developers with clear security guidance.

Creating Security Stories/Tasks:

Don't just say "make it secure." Break security work into concrete Jira tasks:

`[Security] Implement parameterized queries to prevent SQL injection in UserDAO class.`

`[Security] Add input validation using OWASP ESAPI for the registration endpoint.`

`[Security] Set HTTP security headers (CSP, HSTS) in the reverse proxy config.`

3.3 Practice: Security Features & Design

Goal: Build security into architecture.

Secure Design Review Checklist (Questions to ask):

- [ ] How is authentication and authorization handled across components?
- [ ] How are secrets (API keys, passwords) managed? (Use HashiCorp Vault/AWS Secrets Manager, not code/config files).

- [ ] Is data encrypted at rest and in transit?
- [ ] How are logs generated and protected? (Prevent log injection and ensure integrity).

4. Domain 3: SSDL Touchpoints

Focus: Integrating security activities into the software development lifecycle.

4.1 Practice: Architecture Analysis

Goal: Analyze software designs for security flaws.

Lightweight Review Process:

Trigger: Initiated by a Security Champion for any significant new feature or service.

Participants: Lead developer, architect, Security Champion, AppSec engineer.

Duration: 60-minute session using the Threat Modeling process from 3.1.

Output: A list of security tasks added to the sprint backlog.

4.2 Practice: Code Review

Goal: Identify security defects in code.

SAST & Tools Integration Guide:

Select Tools: Choose a SAST tool (e.g., SonarQube, Checkmarx, Semgrep).

Integrate: Run the tool in the CI pipeline. **FAIL THE BUILD** for critical/high-severity findings. This is non-negotiable for automation.

Triage: AppSec team maintains the tool's ruleset, suppresses false positives, and ensures findings are relevant.

Manual Review: Encourage developers to add a `security-review` label for pull requests that they want a Security Champion to look at manually.

4.3 Practice: Security Testing

Goal: Find vulnerabilities in running software.

DAST/IAST in CI/CD Pipeline:

Stage: Pre-Production

Run DAST (e.g., OWASP ZAP, Burp Suite) scans against a staging environment as a nightly build.

Integrate IAST tools (e.g., Contrast Security) into the test environment for real-time feedback during QA testing.

Stage: Production

Schedule regular, automated DAST scans for public-facing apps (scope carefully!).

Use bug bounty platforms (e.g., HackerOne) or contracted pentesters for deeper testing.

5. Domain 4: Deployment

Focus: Protecting software in production and managing security events.

5.1 Practice: Penetration Testing

Goal: Simulate real-world attacks.

Scoping & Bug Triage Workflow:

Scope: Define exactly what is to be tested (e.g., “New payment API v2, staging environment”).

Choose Tester: Internal red team or external vendor.

Provide Access: Give testers documentation, API keys, and a test account.

Triage:

All findings go into a dedicated security bug queue.

AppSec lead assigns a CVSS score and priority.

Development team works on fixes based on priority. Critical bugs must be fixed within a pre-defined SLA (e.g., 72 hours).

5.2 Practice: Software Environment

Goal: Harden the infrastructure.

Hardening Baseline Checklist:

- [ ] Use hardened OS images (e.g., CIS Benchmarks).
- [ ] All infrastructure is defined as code (IaC) and scanned with tools like**Terrascan** or **Checkov** for security misconfigurations *before deployment*.
- [ ] Use WAF (Web Application Firewall) for public web apps.
- [ ] Ensure all communications are encrypted (TLS).

5.3 Practice: Configuration & Vuln Management

Goal: Identify, prioritize, and fix vulnerabilities in deployed apps.

Patch Management SLA:
Define and enforce strict SLAs for fixing vulnerabilities based on severity:

Critical (CVSS >= 9.0): Patch within 24-72 hours.

High (CVSS 7.0 - 8.9): Patch within 1-2 weeks.

Medium (CVSS 4.0 - 6.9): Patch within 1 month.

Low (CVSS < 4.0): Address within next regular release.

Tooling: Use a centralized dashboard (e.g., DefectDojo, ThreadFix) to aggregate findings from SAST, DAST, SCA, and pentests.

6. Putting It All Together: A 90-Day Plan

Phase 1: Quick Wins (First 30 Days)

Start Measuring: Define 3-5 key security metrics.

Automate One Thing: Integrate SCA into CI to block builds with critical lib vulnerabilities.

Train: Identify Security Champions and run one threat modeling workshop.

Scan: Perform a basic DAST scan on your main application.

Phase 2: Process Building (Next 30 Days)

Formalize: Document the threat modeling process and make it mandatory for new features.

Enforce: Integrate SAST into CI and start failing builds for critical findings.

Harden: Create a basic hardening baseline for your cloud environment/IaC.

Define SLAs: Establish vulnerability remediation SLAs with development teams.

Phase 3: Metrics & Optimization (Last 30 Days)

Review Metrics: Analyze your initial metrics. Is MTTR improving? Is test coverage increasing?

Refine Processes: Optimize your threat modeling and code review processes based on feedback.

Expand Scope: Scale the Security Champions program to more teams.

Plan Next Steps: Based on your current state, choose the next set of BSIMM activities to implement.

Appendix: Tools & Automation Ideas

SAST: Semgrep (good for quick wins), SonarQube, Checkmarx, Snyk Code

SCA: Snyk, Mend (WhiteSource)

DAST: OWASP ZAP (free, automatable), Burp Suite Pro

Secrets Management: HashiCorp Vault, AWS Secrets Manager, Azure Key Vault

IaC Scanning: Checkov, Terrascan, TFsec

Vulnerability Management: DefectDojo, ThreadFix

Threat Modeling: OWASP Threat Dragon, IriusRisk

Document Version History:

Version 1.0 (2025-08-30): Initial practical implementation guide compiled by Ivan Piskunov.