

BSIMM



**FOUNDATIONS
REPORT 2022**

TABLE OF CONTENTS

PART 1: EXECUTIVE SUMMARY.....4

TRENDS AND INSIGHTS	5
Why We Do Software Security	5
Where We Do Software Security	5
How We Do Software Security	6

WELCOME TO BSIMM13	6
---------------------------------	---

BSIMM13 DATA HIGHLIGHTS	7
--------------------------------------	---

CALL TO ACTION	8
Plan Your Journey	8
Get a Handle on What You Have	9
Pay Attention to the Latest Trends	9

THE BSIMM SKELETON	10
---------------------------------	----

PART 2: TRENDS AND INSIGHTS.....12

SHIFT EVERYWHERE	13
Translating Risk Numbers into Decisions	13
Continuous Defect Discovery	13
Governance-as-Code	13

SOFTWARE SUPPLY CHAIN RISK MANAGEMENT	13
Software Bill of Materials	14
Open Source Software	14
Vendor Management	14
Training for Outsourced Workers	14

SECURITY INTEGRATION INTO DEVELOPER TOOLCHAINS	14
Dispersal into SDLC vs. Gates	14
Automating Coding Standards	14

EXPANDING SOFTWARE SECURITY	
BEYOND APPLICATIONS AND PRODUCTS	15
Leveraging Operational Data for Continuous Improvement	15
Integration of Knowledge-as-Code	15
Security Champions	15

TOPICS WE'RE WATCHING	15
------------------------------------	----

PART 3: THE BSIMM COMMUNITY..... 16

PARTICIPANTS	17
---------------------------	----

CASE STUDY: LENOVO	18
---------------------------------	----

CASE STUDY: LEADING NORTH AMERICAN FINANCIAL INSTITUTION	19
-----------------------------------------------------------------------	----

CASE STUDY: CRED	19
-------------------------------	----

ACKNOWLEDGEMENTS	20
-------------------------------	----

PART 4: QUICK GUIDE TO SSI MATURITY21

QUICK BASELINE FOR SSI LEADERS	22
Is Your SSI Keeping Pace with Change in Your Software Portfolio?	22
Are You Creating the DevSecOps Culture You Need?	22
Are You Shifting Security Efforts Everywhere in the Engineering Lifecycle?	22
How Does Your SSI Measure Up?	22

USING A BSIMM SCORECARD TO MAKE PROGRESS	23
Understand Your Organizational Mandate	23
Build the Scorecard	23
Make a Strategic Plan and Execute	23

ROLES IN A SOFTWARE SECURITY INITIATIVE	25
------------------------------------------------------	----

PART 5: THE BSIMM FRAMEWORK.....26

CORE KNOWLEDGE	27
-----------------------------	----

UNDERSTANDING THE MODEL	28
--------------------------------------	----

PART 6: THE BSIMM ACTIVITIES29

ACTIVITIES IN THE BSIMM	30
--------------------------------------	----

GOVERNANCE	30
Governance: Strategy & Metrics (SM)	30
Governance: Compliance & Policy (CP)	32
Governance: Training (T)	34

INTELLIGENCE	35
Intelligence: Attack Models (AM)	35
Intelligence: Security Features & Design (SFD)	37
Intelligence: Standards & Requirements (SR)	38

SDLC TOUCHPOINTS	39
SDLC Touchpoints: Architecture Analysis (AA)	39
SDLC Touchpoints: Code Review (CR)	41
SDLC Touchpoints: Security Testing (ST)	42

DEPLOYMENT	44
Deployment: Penetration Testing (PT)	44
Deployment: Software Environment (SE)	44
Deployment: Configuration Management & Vulnerability Management (CMVM)	46

APPENDICES 48

A. ROLES IN A SOFTWARE SECURITY INITIATIVE 49

EXECUTIVE LEADERSHIP	49
SOFTWARE SECURITY GROUP LEADERS	50
SOFTWARE SECURITY GROUP (SSG)	51
SATELLITE (SECURITY CHAMPIONS).....	51
KEY STAKEHOLDERS.....	52

B. HOW TO BUILD OR UPGRADE AN SSI53

STARTING AN SSI: GETTING TO AN EMERGING STATE	53
Create a Software Security Group	54
Document and Socialize the SSDL	55
Inventory Applications in the SSG's Purview	55
Apply Infrastructure Security in Software Environments	55
Deploy Defect Discovery for High-Priority Applications	55
Publish and Promote the Process	56
Progress to the Next Step in Your Journey	56

LESSONS FROM THE COMMUNITY56

Cultures	56
A New Wave in Engineering Culture	57
Understanding More About DevOps.....	58
Convergence as a Goal	58

MATURING AN SSI: HARMONIZING OBJECTIVES59

Establish Leadership and Objectives.....	59
Expand Security Controls	59
Engage Development	59
Inventory and Select In-Scope Software.....	60
Enforce Security Basics Everywhere	60
Integrate Defect Discovery and Prevention	61
Upgrade Incident Response	61
Repeat and Improve.....	61

ENABLING SSIs62

Progress Isn't a Straight Line	62
Push for Agile-Friendly SSIs	62

C. DETAILED VIEW OF THE BSIMM FRAMEWORK 63

THE BSIMM SKELETON.....	63
CREATING BSIMM13 FROM BSIMM12	63
MODEL CHANGES OVER TIME.....	67

D. DATA: BSIMM13 69

AGE-BASED PROGRAM CHANGES.....	69
ACTIVITY CHANGES OVER TIME	71

E. DATA ANALYSIS: VERTICALS 73

IOT, CLOUD, AND ISV VERTICALS	74
FINANCIAL, HEALTHCARE, AND INSURANCE VERTICALS	74
FINANCIAL AND TECHNOLOGY VERTICALS	75
TECHNOLOGY VS. NON-TECHNOLOGY	76
VERTICAL SCORECARDS	76

F. DATA ANALYSIS: LONGITUDINAL 81

BUILDING A MODEL FOR SOFTWARE SECURITY	81
CHANGES BETWEEN FIRST AND SECOND ASSESSMENTS.....	82
CHANGES BETWEEN FIRST AND THIRD ASSESSMENTS	84

G. DATA ANALYSIS: SATELLITE (SECURITY CHAMPIONS)..... 86

H. DATA ANALYSIS: SSG 88

SSG CHARACTERISTICS.....	88
SSG CHANGES BASED ON AGE.....	90



PART 1:
EXECUTIVE
SUMMARY

EXECUTIVE SUMMARY

In 2008, application security, research, and analysis experts set out to gather data on the different paths that organizations take to address the challenges of securing software. Their goal was to conduct in-person interviews with organizations that were known to be highly effective in software security initiatives (SSIs), gather details about their efforts, analyze the data, and publish their findings.

The result was the Building Security In Maturity Model (BSIMM), a descriptive model—published as BSIMM1—that provides a baseline of observed activities (i.e., controls) for software security initiatives (SSIs) to build security in to software and software development. Because these initiatives often use different methodologies and different terminology, the BSIMM also creates a common vocabulary everyone can use. In addition, the BSIMM provides a common methodology for starting and improving SSIs of any size and in any vertical market.

TRENDS AND INSIGHTS

- These BSIMM trends and insights are a distillation of software security lessons learned across 130 organizations that collectively have 11,850 security professionals helping about 410,000 developers do good security work on about 145,000 applications.
- Use this information to inform your own strategy for improvement.

Trends describe shifts in SSI behavior that affect activity implementation across multiple areas. Larger in scope than an activity, or even a capability that combines multiple activities within a workflow, we believe these trends show the way organizations are executing groups of activities within their evolving culture. For example, there's a clear trend toward collecting event-driven security telemetry in addition to (or sometimes even rather than) conducting point-in-time security scans that produce reports people must review. Over time, we've seen a trend in testing being applied earlier in the software lifecycle ("shift left"), followed by trends in additional testing (e.g., composition analysis) and in testing automation (e.g., as checkpoints in the software development lifecycle [SDLC]).

Refer to Part 2 later in this document for more Trends and Insights.

Why We Do Software Security

Software security leaders continue to face pressure to increase the size, scope, depth, and complexity of their SSIs. From government mandates to regulatory changes, technology shifts, budget and hiring constraints, attacker successes, and marketplace demands, software security leaders must do more with less, and do it better. In short, expanded software security governance is a necessity in any modern SSI.

However, governance done with people and checklists alone doesn't scale very far. A trend today is governance-as-code, where security

leaders provide their mandatory requirements (e.g., policies and standards) as checks or guardrails (i.e., as code) in the engineering infrastructure, enabling scaling automatically.

There has been much media, insurance, and executive attention on security issues found in third-party code. This has led to a trend in software supply chain risk management to track and secure external software that's integrated into internal software and systems. To track this trend, BSIMM13 includes a new activity for integrating supply chain risk management.

Open source software is now a common part of nearly every development effort, which has led to a significant increase in efforts around identifying open source and controlling open source risk, whose observation rate—the rate at which we observed this effort in the BSIMM community—grew by nearly 35% from BSIMM12 to BSIMM13. Balancing the cost savings from open source use with the risk incurred is becoming an important governance objective. In some cases, it's possible to control some risk associated with third-party software through contractual terms. We've recently seen a 15% increase in efforts around creating service-level agreement (SLA) boilerplate for software security responsibilities and including these SLAs in vendor contracts.

Finally, software security and the responsibility for security leaders to keep their organizations safe are growing in new ways. Beyond just scaling with software portfolio size, software security is becoming intertwined with cloud security, infrastructure security, container security, orchestration security, site reliability, and much more. These adjacent security disciplines can both support and undermine even the best software security program. We see a trend in bridge-building between these various groups for the purpose of defining and using mutually beneficial security solutions.

Where We Do Software Security

Not so long ago, most organizations were attempting to manage software security risk by doing some testing just before releasing software to production. It quickly became evident that this approach wasn't working and couldn't scale even if it was. The movement to "shift left" in the SDLC put testing earlier, to happen while the code was being written. This was much more scalable and kept uncountable security defects from ever making it to production code. However, such testing—usually static analysis with a tool—was still a time-consuming gate that simply moved the friction between security and engineering from the day before release to production to the day before release to build.

Today, the trend continues toward "shift everywhere," an approach to embedding software security testing throughout the software lifecycle in both development and operations. The move to more testing done more often, usually using smaller tests that run faster, enables the governance-as-code approach that organizations need. Facilitating shift everywhere is a trend to translating risk numbers into decisions where we see more than 25% growth in activities related to combining security testing results to improve decision-making, striving for data-driven change in software security processes, and using metrics to drive resourcing.

As another aspect of shift everywhere, organizations are trending toward distributing testing into engineering workflows—including security tests in QA functional testing automation has grown by almost 50%. There has also been steady growth in use of automated tools, integrating security tools into the QA process, and defining secure deployment parameters and configurations.

Some labor-intensive security testing has been on a downward trend. For example, using secure coding guidance and enforcing secure coding standards have declined for several years. In the past year, however, we've seen a significant spike in both the usage and enforcement aspects, presumably because it's become much easier to enforce coding standards with automated testing rather than with peer code review processes that take up too much valuable development time.

How We Do Software Security

For such a complicated endeavor, software development and its associated security governance was also simple: write some code, build it, then apply all the security testing there was time for. Development then fixed the worst security defects discovered, and some of the remainder became requirements for the next release. Today, important aspects of software security are embedded throughout people, process, technology, and culture.

Doing testing at a single SDLC gate became unacceptably inadequate over time, and today we see a trend toward continuous defect discovery, especially testing that can be automated into lifecycle tooling. For example, effort in the BSIMM Code Review and Security Testing practices each grew at almost twice the rate of effort in the Penetration Testing and Architecture Analysis practices. There is also continued growth in monitoring automated asset creation, with over half the total observations occurring in the past year.

Doing good software security requires accurate and comprehensive knowledge of software assets beyond compiling the list of software developed in-house. A rapidly growing trend is to go beyond a simple application list and to track all components within all deployed code. Here, we've seen a nearly 30% growth in efforts related to creating software bills of materials (SBOMs), which improve the software inventory and help with incident response.

Good software security also requires a trained workforce. The trend in providing software security training for vendors and outsourced software workers increased steadily for years but recently dropped by 30%. It's possible organizations have found other ways to ensure that software suppliers have adequate software security programs, perhaps through a combination of attestations, reviews, and contractual agreements.

We've seen a nearly 30% growth in efforts related to creating software bills of materials (SBOMs), which improve the software inventory and help with incident response.

Data turned into decision support knowledge is the lifeblood of a risk management program. Organizations are, for example, leveraging operational data about security defects to look for and fix all occurrences of important defects found in operations, with a recent growth of 175% in this effort. They are also using this decision support knowledge to improve the SDLC based on issues found in operations, which has grown by 70%, and using SDLC knowledge to improve policy, which has grown by over 80%. This, and other useful knowledge, is also beginning to be turned into code when possible. Efforts to define secure deployment parameters and configurations (and to use application containers to support security goals) grew by nearly 20%, and the use of orchestration for containers grew by nearly 30%.

Once again, labor-intensive efforts are hard to staff, making them hard to scale. Security champions—a team of people skilled in various aspects of software security—are a good way to ensure that evangelism, training, and governance reaches all parts of the organization. There has recently been a 15% increase in the number of firms that have a security champions group. Note also that there has been a continuous trend over the years in organizations with a champions group scoring higher than organizations without, currently at about 35% (13 points) higher on average for BSIMM13.

WELCOME TO BSIMM13

— If you're in charge of an SSI, understanding the BSIMM model and its use by the community will help you plan strategic improvements. If you're running technical aspects of an initiative, you can use the how-to guide (in Part 4) and activity descriptions (in Part 6) to help define tactical improvements to people, process, technology, and culture.

Each BSIMM annual report is the result of studying real-world SSIs, which some organizations refer to as their application security program or product security program, or as their DevSecOps effort. Each year, a variety of firms in different industry verticals use the BSIMM to create a software security scorecard for their programs that they then use to manage their SSI improvements. Here, we present BSIMM13 as built directly out of the data we observed in 130 firms.

In the rapidly changing software security field, it's important to understand what other organizations are doing in their SSIs. Comparing the efforts of hundreds of companies to your own will directly inform your strategy for your own software security efforts.

The BSIMM core knowledge is the activities we directly observed in the community—the group of firms that participate in using the BSIMM as part of their SSI management. Each community member has their own unique SSI with an emphasis on the build-security-in activities important to their business objectives, but they collectively use the activities captured here. We organize that core knowledge into a software security framework (SSF), represented in Figure 7. The SSF is organized into four domains—Governance, Intelligence, SSDL Touchpoints, and Deployment—with those domains currently embracing 125 activities. The Governance domain, for example, includes activities that fall under the organization, management, and measurement practices of an SSI.

Descriptions of the BSIMM domains, practices, and activities can be found at www.bsimm.com/framework.html.

From an executive perspective, you can view BSIMM activities as controls implemented in a software security risk management framework. The implemented activities might function as preventive, detective, corrective, or compensating controls. Positioning the activities as controls allows for easier understanding of the BSIMM's value by governance, risk, compliance, legal, audit, and other risk management groups.

BSIMM Terminology

Nomenclature has always been a problem in computer security, and software security is no exception. Several terms used in the BSIMM have particular meaning for us. The following list highlights some of the most important terms used throughout this document:

- **Activity.** Actions or efforts carried out or facilitated by the SSG as part of a practice. Activities are divided into three levels in the BSIMM based on observation rates.
- **Capability.** A set of BSIMM activities spanning one or more practices working together to serve a cohesive security function.
- **Champions.** Interested and engaged developers, cloud security engineers, deployment engineers, architects, software managers, testers, and people in similar roles who have an active interest in software security and contribute to the security posture of the organization and its software.
- **Community.** The group of firms in the current data pool.
- **Data pool.** The collection of assessment data from the current community.
- **Domain.** One of the four categories the framework is divided into, i.e., Governance, Intelligence, SSDL Touchpoints, and Deployment.
- **Practice.** A grouping of BSIMM activities. The SSF is organized into 12 practices, three in each of four domains.
- **Satellite.** A group of individuals, often called security champions, that is organized and leveraged by an SSG.
- **Secure SDL (SSDL).** Any software lifecycle with integrated software security checkpoints and activities.
- **Software security framework (SSF).** The basic structure underlying the BSIMM, comprising 12 practices divided into four domains.
- **Software security group (SSG).** The internal group charged with carrying out and facilitating software security. The group's name might also have an appropriate organizational focus, such as application security group or product security group.
- **Software security initiative (SSI).** An organization-wide program to instill, measure, manage, and evolve software security activities in a coordinated fashion. Also referred to in some organizations as an application security program, product security program, or perhaps as a DevSecOps program.

As with any research work, there are some terms that have specific meanings in the BSIMM. The box below shows the most common BSIMM terminology.

BSIMM13 DATA HIGHLIGHTS

- Use the information in this section to answer common questions about BSIMM data, such as, “What are some community statistics?” “Which activities are most firms doing?” and “How are software security efforts changing over time?”

Activities are the building blocks and smallest unit of granularity that are implemented across organizations to build SSIs. Rather than dictating a set of prescriptive activities, the purpose of the BSIMM is to descriptively observe and quantify the actual activities carried out by various kinds of SSIs across many organizations.

The BSIMM is an observational model that reflects current software security efforts, so we adjust it annually to keep it current. For BSIMM13, we've made the following changes to the model based on what we see in the BSIMM community:

- We moved activities related to controlling open source risk, implementing cloud security controls, hosting software security events, and requiring an annual training refresher because we now see them more frequently.
- We moved activities related to security experts leading design review efforts and using centralized defect reporting for targeted training because they're growing much more slowly than other common activities in their practice areas.
- We added the following activities because we are beginning to see them more in the community:
 - Integrate software supply chain risk management
 - Perform application composition analysis on code repositories
 - Do attack surface management for deployed applications

Unique in the software security industry, the BSIMM project has grown from nine participating companies in 2008 to 130 in 2022, now with nearly 3,350 software security group (SSG) members and over 8,500 satellite members (aka security champions). The average age of the participants' SSIs is 5.0 years. The BSIMM project shows consistent growth even as participants enter and leave the community over time—we added 27 firms for BSIMM13 and dropped 25 whose data hadn't been refreshed.

This 2022 edition of the BSIMM report—BSIMM13—examines anonymized data from the software security activities of 130 organizations across various verticals, including cloud, financial services, financial technology (FinTech), independent software vendors (ISVS), insurance, Internet of Things (IoT), healthcare, and technology organizations.

The popular business book, *The 7 Habits of Highly Effective People*, explores the theory that successful individuals share common qualities in achieving their goals and that these qualities can be identified and applied by others. The same premise can be applied to SSIs. Listed in

Table 1 are the 10 most observed activities in the BSIMM13 data pool. The data suggests that if your organization is working on its own SSI, you should consider implementing these activities.

Table 2 shows some activities that have experienced exceptionally high growth over the past 12 months. Not surprisingly, some of these activities, such as *control open source risk* and *identify open source*, are mentioned in the Trends and Insights section. In addition, the activity introduced in BSIMM12, *streamline incoming responsible vulnerability disclosure*, has the largest increase in observation count. Note that for some of the activities in Table 2, the growth in observation is a new change. For example, the activities *make code review mandatory for all projects*, *create a security portal*, and *provide expertise via open collaboration channels* saw virtually no growth in the previous three years but all saw a significant jump in observation rates in the last 12 months. While one year of new data is not sufficient to establish a trend, it is worth paying attention to and considering for your program.

CALL TO ACTION

- Use the information in this section to prioritize improvements in your SSI and perhaps also in the SSIs of your most important software suppliers and partners.

Every SSI has room for improvement, whether it's improving scale, effectiveness, depth, risk management, the framework of deployed activities, resourcing, or anything similar. The following suggestions represent the broad efforts we see happening in the BSIMM community, and various parts are likely right for your program as well.

Take stock of your SSI. It's important to periodically look at your program through a different lens.

Plan Your Journey

- Take stock of your SSI. It's important to periodically look at your program through a different lens, and the BSIMM enables that. Use the guidance in Part 4 to create your own SSI scorecard and compare it to your expectations.
- Create a vision and strategic plan. Use the activity descriptions in Part 6 when creating a prioritized action plan for business areas where your current SSI efforts fall short. Typical investment areas include risk management, digital transformation, technical debt removal, technology insertion, and process improvement.

Get a Handle on What You Have

- Inventory all your code. It's likely that you'll need specialized automation to keep track of all the code you write and all the code

BSIMM13 TOP 10 ACTIVITIES	
PERCENT	DESCRIPTION
90.0%	Implement security checkpoints and associated governance.
88.5%	Ensure host and network security basics are in place.
88.5%	Identify privacy obligations.
87.7%	Create or interface with incident response.
87.7%	Use external penetration testers to find problems.
86.9%	Perform security feature review.
83.1%	Perform edge/boundary value condition testing during QA.
82.3%	Use automated code review tools.
80.0%	Integrate and deliver security features.
79.2%	Translate compliance constraints to requirements.

TABLE 1. TOP ACTIVITIES BY OBSERVATION PERCENTAGE. The most frequently observed activities in BSIMM13 are likely important to all SSIs.

BSIMM13 TOP 10 ACTIVITIES GROWTH BY COUNT	
INCREASE	DESCRIPTION
20	Streamline incoming responsible vulnerability disclosure.
20	Implement cloud security controls.
18	Control open source risk.
18	Identify open source.
16	Create a standards review process.
15	Gather and use attack intelligence.
13	Provide expertise via open collaboration channels.
13	Make code review mandatory for all projects.
13	Create a security portal.
11	Schedule periodic penetration tests for application coverage.

TABLE 2. TOP ACTIVITIES BY RECENT GROWTH IN OBSERVATION COUNT. These activities had the largest growth in BSIMM13, out of 44 firms measured during the last 12 months, which means they are likely important to your program now or will be soon.

you bring in from outside the organization. A simple application inventory will be useful for some things, such as naming risk managers, but you'll quickly need specialized inventories such as BOMs, API and microservices lists, code that is subject to specific compliance needs, and much more.

- Automate, automate, automate. Search for ways to eliminate error-prone manual processes and reduce friction between governance and engineering groups, including automating security decisions. This will require some policy-as-code effort and tools integration, and maybe even bringing development skills into the SSG.
- Gather all the data. As more processes become code and more policy and standards become machine-readable, day-to-day development and operations will generate significantly more telemetry about what's happening and why. Use this data to ensure that everything's working as expected.

Even perfect software can have its security undermined by mistakes

Here are some suggestions on reading through this BSIMM report:

- If you're experienced with the BSIMM, or if you need some content to help make your case with executive management, then Part 2: Trends and Insights are probably what you're looking for.
- If this is your first time with the BSIMM, we recommend first reading Part 5 for context and then returning here to decide what to read next.
- If you're starting an SSI or an SSG, or looking to mature an existing program, start with Part 4: Quick Guide to SSI Maturity, then move to Appendix B: How to Build or Upgrade an SSI, and then read through the activities in Part 6.
- If you want to get right into the types of software security controls organizations are using in their SSIs, or if you are working on building out capabilities, then read Part 6: The BSIMM Activities.
- If you want to see a summary of the BSIMM13 data, review Appendix D.
- If you want to look at our analysis of the BSIMM data, review Appendices E through H.
- If you'd like to see a brief case study, the Case Studies section has you covered.

elsewhere in the organization.

Pay Attention to the Latest Trends

- Innovate in digital transformation. Encourage your SSG and other security stakeholders to experiment with ways to deliver security value directly into engineering processes, especially where current security testing tools don't always keep up with engineering changes, such as with serverless architectures, single-page applications, API security, and zero trust.
- Secure the software supply chain. Nearly every organization today uses third-party code and provides code as a third party to other organizations. While producing SBOMs is easy, the management of software, SBOMs, vendors, and vulnerability information is much more complicated.
- Expand software security into adjacencies. Even perfect software can have its security undermined by mistakes elsewhere in the organization. Make explicit ties between the SSI and other security stakeholders working in areas such as container security, orchestration security, cloud security, infrastructure security, and site reliability.

In summary, the data shows that new SSIs—from just started to 18 months old—are typically doing about 33 BSIMM activities. These organizations are also beginning to scale these activities across their software portfolio, deal with all the change going on around them, and evolve their risk management strategy.

THE BSIMM SKELETON

The BSIMM skeleton provides a way to view activities at a glance, which is useful when thinking about your own SSI. The skeleton is shown in Figure 1, organized by domains and practices. More complete descriptions of the activities and examples are available in Part 6 of this document.

- Use this skeleton to understand the software security activities included in BSIMM13. A list of software security controls can be a very helpful guide, and the BSIMM project has worked since 2008 to ensure that its content matches real-world efforts.

GOVERNANCE		
STRATEGY & METRICS	COMPLIANCE & POLICY	TRAINING
<ul style="list-style-type: none"> Publish process and evolve as necessary. Educate executives on software security. Implement security checkpoints and associated governance. Publish data about software security internally and use it to drive change. Enforce security checkpoints and track exceptions. Create or grow a satellite (security champions). Require security sign-off prior to software release. Create evangelism role and perform internal marketing. Use a software asset tracking application with portfolio view. Make SSI efforts part of external marketing. Identify metrics and use them to drive resourcing. Integrate software-defined lifecycle governance. Integrate software supply chain risk management. 	<ul style="list-style-type: none"> Unify regulatory pressures. Identify privacy obligations. Create policy. Build a PII inventory. Require security sign-off for compliance-related risk. Implement and track controls for compliance. Include software security SLAs in all vendor contracts. Ensure executive awareness of compliance and privacy obligations. Document a software compliance story. Ensure compatible vendor policies. Drive feedback from software lifecycle data back to policy. 	<ul style="list-style-type: none"> Conduct software security awareness training. Deliver on-demand individual training. Include security resources in onboarding. Enhance satellite (security champions) through training and events. Create and use material specific to company history. Deliver role-specific advanced curriculum. Host software security events. Require an annual refresher. Reward progression through curriculum. Provide training for vendors and outsourced workers. Provide expertise via open collaboration channels. Identify new satellite members (security champions) through observation.

INTELLIGENCE		
ATTACK MODELS	SECURITY FEATURES & DESIGN	STANDARDS & REQUIREMENTS
<ul style="list-style-type: none"> Use a data classification scheme for software inventory. Identify potential attackers. Gather and use attack intelligence. Build attack patterns and abuse cases tied to potential attackers. Create technology-specific attack patterns. Maintain and use a top N possible attacks list. Collect and publish attack stories. Build an internal forum to discuss attacks. Have a research group that develops new attack methods. Create and use automation to mimic attackers. Monitor automated asset creation. 	<ul style="list-style-type: none"> Integrate and deliver security features. Application architecture teams engage with the SSG. Leverage secure-by-design components and services. Create capability to solve difficult design problems. Form a review board or central committee to approve and maintain secure design patterns. Require use of approved security features and frameworks. Find and publish secure design patterns from the organization. 	<ul style="list-style-type: none"> Create security standards. Create a security portal. Translate compliance constraints to requirements. Create a standards review process. Identify open source. Create SLA boilerplate. Control open source risk. Communicate standards to vendors. Use secure coding standards. Create standards for technology stacks.

SSDL TOUCHPOINTS		
ARCHITECTURE ANALYSIS	CODE REVIEW	SECURITY TESTING
<ul style="list-style-type: none"> • Perform security feature review. • Perform design review for high-risk applications. • Use a risk methodology to rank applications. • Perform architecture analysis using a defined process. • Standardize architectural descriptions. • Have SSG lead design review efforts. • Have engineering teams lead AA process. • Drive analysis results into standard architecture patterns. • Make the SSG available as an AA resource or mentor. 	<ul style="list-style-type: none"> • Perform opportunistic code review. • Use automated code review tools. • Make code review mandatory for all projects. • Assign code review tool mentors. • Use custom rules with automated code review tools. • Use a top N bugs list (real data preferred). • Use centralized defect reporting to close the knowledge loop. • Build a capability to combine AST results. • Create capability to eradicate bugs. • Automate malicious code detection. • Enforce secure coding standards. 	<ul style="list-style-type: none"> • Perform edge/boundary value condition testing during QA. • Drive tests with security requirements and security features. • Integrate opaque-box security tools into the QA process. • Drive QA tests with AST results. • Include security tests in QA automation. • Perform fuzz testing customized to application APIs. • Drive tests with design review results. • Leverage code coverage analysis. • Begin to build and apply adversarial security tests (abuse cases). • Implement event-driven security testing in automation.

DEPLOYMENT		
PENETRATION TESTING	SOFTWARE ENVIRONMENT	CONFIGURATION MANAGEMENT & VULNERABILITY MANAGEMENT
<ul style="list-style-type: none"> • Use external penetration testers to find problems. • Feed results to the defect management and mitigation system. • Use penetration testing tools internally. • Penetration testers use all available information. • Schedule periodic penetration tests for application coverage. • Use external penetration testers to perform deep-dive analysis. • Customize penetration testing tools. 	<ul style="list-style-type: none"> • Use application input monitoring. • Ensure host and network security basics are in place. • Implement cloud security controls. • Define secure deployment parameters and configurations. • Protect code integrity. • Use application containers to support security goals. • Use orchestration for containers and virtualized environments. • Use code protection. • Use application behavior monitoring and diagnostics. • Create bills of materials for deployed software. • Perform application composition analysis on code repositories. 	<ul style="list-style-type: none"> • Create or interface with incident response. • Identify software defects found in operations monitoring and feed them back to development. • Have emergency response. • Track software bugs found in operations through the fix process. • Develop an operations software inventory. • Fix all occurrences of software bugs found in operations. • Enhance the SSDL to prevent software bugs found in operations. • Simulate software crises. • Operate a bug bounty program. • Automate verification of operational infrastructure security. • Publish risk data for deployable artifacts. • Streamline incoming responsible vulnerability disclosure. • Do attack surface management for deployed applications.

FIGURE 1. THE BSIMM SKELETON. Within the SSF, the 125 activities are organized into the 12 BSIMM practices, which are within four domains.



PART 2:
TRENDS AND
INSIGHTS

TRENDS AND INSIGHTS

Where do we get the data, and why should you participate?

- BSIMM data originates in interviews conducted with member firms during a BSIMM assessment. Through these in-depth conversations, assessors look for the existence of BSIMM activities and assign credit for activities that are performed with sufficient coverage across the organization, formality to be repeatable and consistent, and depth to be effective at managing associated risk. After each assessment, the observation data is anonymized and added to the BSIMM data pool, where statistical analysis is performed to highlight trends in how firms secure their software. You can use this information to understand what others are doing to then inform your own strategy.

Businesses have seen drivers, pressures, and threats come from nearly every direction. Ransomware attacks have put pressure on supply chains and manufacturing sectors. Industries are facing new regulations or having to re-adapt to old ones as cryptocurrencies go mainstream and interact with currency and banks. The US government has made cybersecurity a priority and is releasing executive orders to put it on industry's radar. The outbreak of war is bringing hacktivism back after a relatively quiet period. Of course, there have also been new software languages and technology stacks, massive shifts to the cloud, and over two years of working from home.

These and other external drivers are being met by organizational transformations facilitated by new technologies, expanded processes, changes in the security culture, and increased supply chain security efforts.

SHIFT EVERYWHERE

Starting more than 15 years ago, the “shift left” trend drove firms to focus on moving security testing earlier in the development process, starting with doing SAST during development. Much more recently, “shift everywhere” extends this trend into making security testing continuous throughout the software lifecycle. This means that smaller, faster, pipeline-driven security tests are conducted as soon as there is an artifact to test. These tests are often smaller and context-specific, such as validating the use of a required library during a pull request, rather than waiting until a build cycle or a penetration test—and these tests can happen anywhere from design to production. A shift everywhere approach is useful for more than just testing for vulnerabilities in a timely fashion; it also facilitates automating governance checks and measuring risk in various parts of the software lifecycle. For example, shifting right into production might entail using automated tests to continuously verify that only those APIs with proper documentation are allowed to receive certain traffic.

Translating Risk Numbers into Decisions

BSIMM13 data shows an increase in firms that collect and combine data from various sources throughout the SDLC to support security decisions. There was more than 25% average growth in related activities such as *build a capability to combine AST results, publish data about software security internally and drive change*, and *identify metrics and use them to drive resourcing*. Collecting and combining data is an important step in shaping risk-based secure SDLC governance and is also a necessary step in governance-as-code efforts.

Continuous Defect Discovery

Firms are increasing adoption of automated defect discovery approaches that favor continuous monitoring and reporting over expert-intensive point-in-time defect discovery. For example, effort in the Code Review and Security Testing practices each grew at almost twice the rate of the effort in the Penetration Testing and Architecture Analysis practices. There is also continued growth in *monitoring automated asset creation*, with over half the total observations occurring in the past year. Shifting everywhere in the SDLC with integrated tooling is an important step in increasing governance while minimizing friction with engineering processes.

Governance-as-Code

Firms are augmenting their implementations of governance-as-code by integrating off-the-shelf CI/CD pipeline solutions with their custom automation or in-house solutions. In some cases, decisions about what to test and when are being implemented in the same commercial software that runs integration tests and software pushes. These automation approaches enable the translation of software security standards and policy into human-readable configuration code or simplified code that conducts software vulnerability discovery.

SOFTWARE SUPPLY CHAIN RISK MANAGEMENT

Increased media attention on critical vulnerabilities discovered in third-party libraries and on supply chain instability caused by global events has increased executive attention on software risk that doesn't originate in the firm's own SDLC. Efforts to manage software supply chain risk are focusing on tracking and securing software that is integrated into in-house-built software and ensuring that software suppliers follow best practices.

After observing increased efforts around controlling risk associated with software brought into the firm, BSIMM13 added the *integrate supply chain risk management* activity. This activity captures efforts to manage risk through governance-driven access and usage controls, maintenance standards, and provenance data. Because software supply chain risk can be introduced at any point in the lifecycle of internally built or bespoke software, firms are moving to automated solutions to ensure that all access, usage, and modifications are done in accordance with policy everywhere in the software lifecycle.

Software Bill of Materials

An SBOM is a machine-readable listing of all the components included in a set of software and aids in identifying which software components could include a publicly disclosed security vulnerability. Firms are adding SBOM generation to their security capabilities to both aid in managing the risk posed by vulnerabilities discovered in the open source they use and to improve their ability to respond promptly to disclosed vulnerabilities. This usefulness likely drove the 30% growth of *create bills of materials for deployed software*. This new effort also contributed to the addition of a new activity for BSIMM13: *perform application composition analysis on code repositories*.

Open Source Software

Firms are getting better at managing the risk associated with integrating open source software into their own applications. Increased media coverage of vulnerabilities found in open source libraries is bringing added executive attention to this area, and the availability of software composition analysis tooling is continuing to fuel year-over-year growth of the *identify open source* and *control open source risk* activities, which grew by nearly 35%. Firms won't be able to get ahead of critical vulnerabilities in their open source libraries without building a comprehensive approach to managing this risk.

Vendor Management

Firms are increasing pressure on vendors by communicating and imposing software security standards on the supplied software. Observations of the *communicate standards to vendors* and *ensure compatible vendor policies* activities grew by over 50% in BSIMM13. Organizations are also increasing their use of standard SLA terms in contracts with vendors and outsource providers to ensure that third-party software won't jeopardize compliance with their software security standards. The *create SLA boilerplate* and *include software security SLAs in all vendor contracts* activities are continuing to grow year-over-year and saw a 15% increase in observations. Ensuring that vendor-supplied software is held to the same security standards as internally built software is essential to managing risk across the entire software portfolio.

Training for Outsourced Workers

Not all trends happen in a positive direction. The activity with the largest drop in observation rates in BSIMM13 is *provide training for vendors and outsourced workers*. The observations for this activity grew steadily over the lifetime of the BSIMM. In BSIMM13, however, the observation rate fell by 30%. Of the 44 firms measured between BSIMM12 and BSIMM13, only two were providing training to vendors and outsourced workers, and the overall BSIMM13 measurement fell to 16 observations as data aged out of the BSIMM13 data pool. This fall in observations might also be linked to growth in the *create SLA boilerplate* and *include software security SLAs in all vendor contracts* activities, where organizations might specify training requirements that contracted firms are expected to provide their development teams.

SECURITY INTEGRATION INTO DEVELOPER TOOLCHAINS

Developers and software vendors continue to make progress in integrating security options into CI/CD pipelines and toolchains. These integrations provide faster and tighter processes that reduce friction, improve coverage, and make the shift everywhere concept a reality.

In the early days of application security, firms found vulnerabilities everywhere they looked—in production, in pre-release products, and in news reports about their software. Shift left was a call to move testing efforts earlier into the development lifecycle to find and fix software vulnerabilities before they could be taken advantage of in production. For a waterfall development structure, shift left meant examining designs during the design phase, testing code when that code was being built, and testing applications as soon as they could run. This view had to evolve and adapt as Agile sprints meant, for example, that the design phase might last for an hour every four weeks, and the shift left team might not get the recurring meeting invite.

Shift everywhere requires a modernized testing philosophy that uses smaller, faster, sometimes pipeline-driven tests to look for issues whenever there is an opportunity to check software and automated processes for verifying adherence to security expectations.

Dispersal into SDLC vs. Gates

Firms are shifting responsibility from monolithic permit-to-release gates into smaller automated checks embedded within the SDLC—one example is the *include security tests in QA automation* activity growing by nearly 50% in BSIMM13. The commonly performed automated testing activities *use automated code review tools*, *integrate opaque-box security tools into the QA process*, and *define secure deployment parameters and configurations* continued their above average growth, as compared to the lack of growth in activities associated with penetration testing and manual code review. Firms can ease the friction associated with enforcing security policy by applying the right test at the right time, such as by incorporating security testing into QA and other automated checkpoints within the SDLC.

Automating Coding Standards

The activities around generating and enforcing coding standards have traditionally been among the rarer activities. Observations of the *use secure coding standards* activity declined by more than 60% from BSIMM6 to BSIMM12. Similarly, firms have had trouble mandating those same standards, with *enforce secure coding standards* dropping to zero in BSIMM12. But in BSIMM13, the observation rate for *use secure coding standards* grew by almost 90%, and there are three new observations for *enforce secure coding standards*. Firms are perhaps finding ways to move from labor-intensive effort to automating standards enforcement via, for example, security anti-patterns that are automatically tested for with pipeline-driven checks. This rebound indicates firms are changing how they use and enforce coding standards by taking advantage of improved automation support in development toolchains.

EXPANDING SOFTWARE SECURITY BEYOND APPLICATIONS AND PRODUCTS

Application security teams have had to follow whenever software security has expanded to include infrastructure-as-code, containers, cloud platforms, and more. The key to keeping up with new security domains is redefining how the SSI is positioned within the firm. It's no longer enough to have a hands-off, test-and-enforce approach to security. Instead, SSI leaders need to become thought leaders, influencers, mentors, and enablers of their peers to ensure that infrastructure is built securely, APIs have the needed controls to securely communicate across new architectures, and application security is proactive in preventing risk in the first place.

Leveraging Operational Data for Continuous Improvement

There is tremendous growth in activities that indicate security teams are working with operations to secure the application portfolio. When a vulnerability is found in operations, the first step is to work with developers to fix that vulnerability. To go beyond that, some firms are also using that bug to start a wider process, as seen in observations of the *fix all occurrences of software bugs found in operations* activity growing by 175%. The next step in continuous security improvement goes beyond fixing and into discovering why the bugs were introduced, then building a capability to prevent the error, as shown by the *enhance the SSDL to prevent software bugs found in operations* activity growing by over 70%. Finally, observations of the *drive feedback from software lifecycle data back to policy* activity grew by over 80%, showing that firms are learning from these steps and updating policy based on expanded bug eradication efforts.

Integration of Knowledge-as-Code

SSGs are working with infrastructure teams to capture security knowledge and encode it in human-readable, machine-deployable configurations. This partnership allows developers to have safe and reliable production platforms for their software to run on, and here we see that observed counts of the *define secure deployment parameters and configurations* and *use application containers to support security goals* activities both grew by nearly 20%. Observations of *leverage secure-by-design components and services* also grew by nearly 20% as firms built a library of reusable and vetted security IP. In addition, firms took advantage of improved infrastructure automation and orchestration to deploy applications in containers that are monitored for configuration drift and non-compliance, which contributed to a nearly 30% growth in observations of the *use orchestration for containers and virtualized environments* activity.

Security Champions

A perennial trend is that firms with a security champions (satellite) group score, on average, 35% (13 points) higher than firms without one. A security champions group allows the SSG to have deputized security experts embedded throughout the SDLC, which enables firms to get broader and deeper coverage of security activities across their software portfolio. This engagement has historically been crucial to executing security tasks such as integrating tools, remediating security defects, responding to security incidents, offering just-in-time training, and motivating good security practices in application teams.

As organizations seek to integrate security into their development and operations, security champions continue to drive some DevSecOps transformation at the developer level. The presence of security champion programs grew by 15% in BSIMM13.

TOPICS WE'RE WATCHING

In this industry, the only constant is change. Continuing innovation in architectures, hosting environments, development languages, attackers, attacks, and even new market areas all contribute to the ongoing changes in what SSIs are expected to secure, how they go about securing them, and how developers can take advantage of these advances in a secure fashion.

The following topics might influence future trends and BSIMM activities:

- **API security and visibility.** All firms are struggling with API documentation, given that undocumented API endpoints aren't easily discoverable, securable, or testable and can be labor-intensive to document once they are discovered.
- **Smart contracts and blockchains.** The blockchain with smart contracts is changing the way companies execute agreements and interact with clients and the world.
- **Automated SDLC observability platforms.** In order to help organizations combine, normalize, and make sense of data from disparate sources, observability platforms seek to automatically collect, process, and report data about all facets of application development, testing, and operations.
- **Zero trust.** While not a new idea, zero trust is now coming into its own as the architecture that might solve the porous attack surface represented by cloud-hosted, microservice-driven, containerized API architectures.
- **AI-generated code.** As we hear stories about complex code writing other complex code, we look forward to learning how that impacts SSIs.



PART 3:
THE BSIMM
COMMUNITY

THE BSIMM COMMUNITY

- The BSIMM community comprises software security leaders and team members from around the globe.
- They have a common mission to continuously improve their SSIs in light of changes in the world around them.
- You can use this information to learn from their efforts.

This 2022 edition of the BSIMM report—BSIMM13—examines anonymized data from the software security activities of 130 organizations. This diverse group spans multiple sizes of security teams, development teams, and software portfolios, as well as regions, vertical markets, and security team ages.

PARTICIPANTS

The 130 organizations in BSIMM13 fall across various verticals, including cloud, financial services, FinTech, ISVS, insurance, IoT, healthcare, and technology organizations (see Figure 2). They also fall across multiple regions.

Unique in the software security industry, the BSIMM project has grown from nine participating companies in 2008 to 130 in 2022, currently with nearly 3,350 software security group members and over 8,500 satellite members (aka security champions). Today, the average age of the participants' SSIs is 5.0 years. As seen in Table 3, the BSIMM project shows consistent growth even as participants enter and leave the community over time.

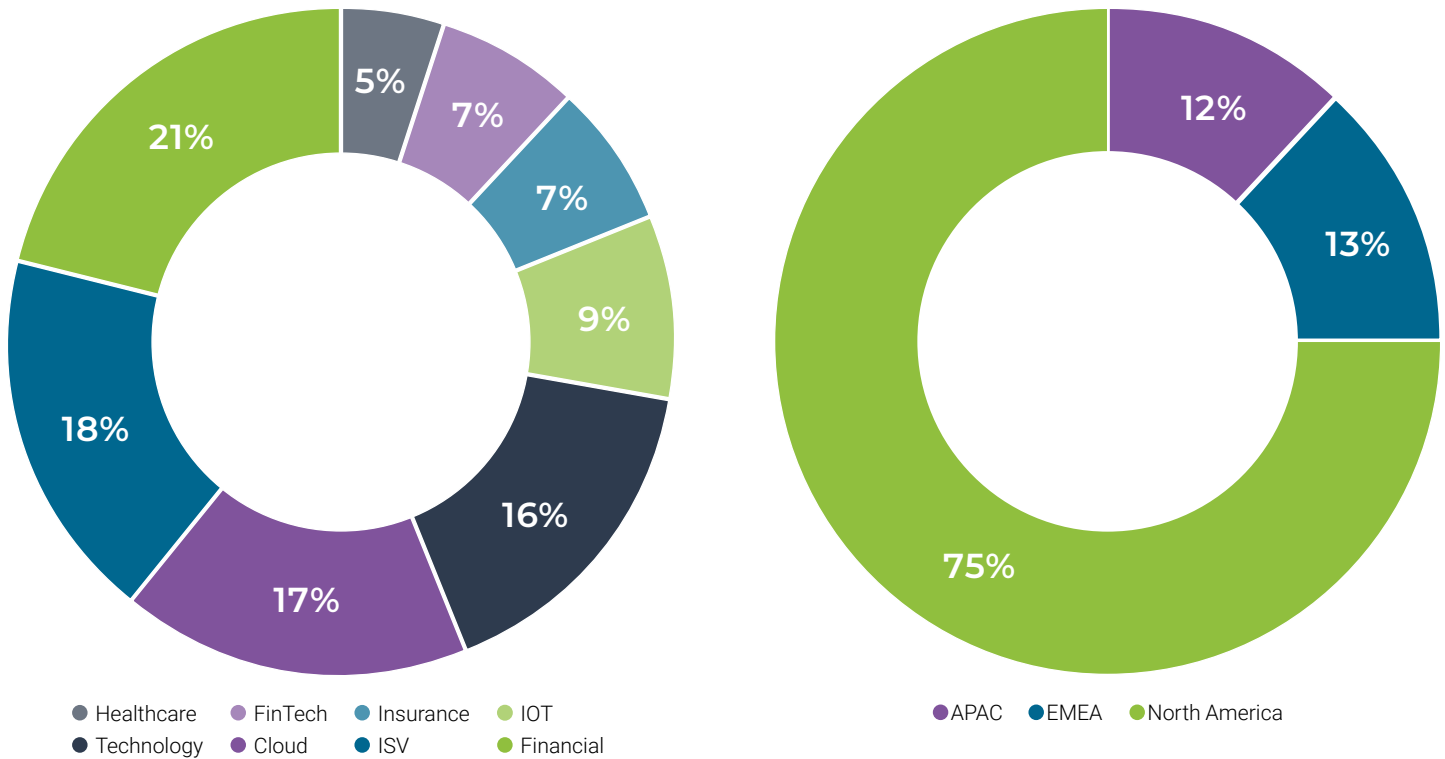


FIGURE 2. BSIMM13 COMMUNITY PARTICIPANTS. Participant percentages per tracked vertical and region.

BSIMM COMMUNITY NUMBERS OVER TIME								
	BSIMM13	BSIMM12	BSIMM11	BSIMM10	BSIMM9	BSIMM8	BSIMM7	BSIMM1
Firms	130	128	130	122	120	109	95	9
SSG Members	3,342	2,837	1,801	1,596	1,600	1,268	1,111	370
Satellite Members	8,508	6,448	6,656	6,298	6,291	3,501	3,595	710
Developers	408,999	398,544	490,167	468,500	415,598	290,582	272,782	67,950
Applications	145,303	153,519	176,269	173,233	135,881	94,802	87,244	3,970
Average SSG Age (Years)	5.00	4.41	4.32	4.53	4.13	3.88	3.94	5.32
SSG Average of Averages (SSG per Developers)	3.01 / 100	2.59 / 100	2.01 / 100	1.37 / 100	1.33 / 100	1.60 / 100	1.61 / 100	1.13 / 100

TABLE 3. BSIMM COMMUNITY NUMBERS OVER TIME. The chart shows how the BSIMM study has grown over the years.

CASE STUDY: LENOVO

“The BSIMM helps Lenovo plan and measure our own security program and gain a sense of the practice areas that are most important to our customers”

Lenovo is a \$70 billion revenue global technology powerhouse, ranked #159 in the Fortune Global 500, employing more than 70,000 people around the world, and serving millions of customers in 180 markets. Focused on a bold vision to deliver smarter technology for all, Lenovo has built on its success as the world’s leading PC player by expanding into new growth areas of infrastructure, mobile, solutions and services.

Securing Lenovo Products

“I lead the Lenovo Infrastructure Solutions Group (ISG) Product Security Office and am part of Lenovo’s corporate security leadership team,” says Bill Jaeger, Executive Director of Lenovo’s ISG Product Security Office. “My organization is responsible for Lenovo ISG’s secure development life cycle and securing the products Lenovo ISG sells, such as data center and edge servers, high-performance (super) computers, enterprise storage, system management software, and other infrastructure solutions.”

“Our product security group covers the entire portfolio of products and productized services – from edge to cloud – including firmware, mobile applications, virtual appliances, traditional on-prem applications and utilities, hosted applications, and ‘as-a-Service’ offerings. We perform hundreds of security assessments and reviews each year encompassing all new product introductions and releases of Lenovo-branded ISG product and productized service offerings.”

“Lenovo is committed to offering products that meet or exceed industry standards for security. Our customers must be able to use Lenovo’s products with confidence that they have the tools that enable them to protect their data, and that our products minimize the risk of vulnerability to malicious or unauthorized use or attack by any third party.”

Key principles of Lenovo’s software security group—established as a corporate product security program in 2014—include:

- Security is designed into Lenovo products
- Lenovo has robust security processes
- Security is considered throughout the product lifecycle
- Employees, stakeholders, and suppliers are required to support these commitments

- Lenovo product security program establishes Product Security Offices led by a Chief Security Officer
- “We’ve seen a measurable reduction in product security issues reported to our Product Security Incident Response Team for newer products as compared to legacy products,” Jaeger continues. “For example, reductions in product vulnerability count and severity as well as increased product security resiliency over time. Product security translates into enhanced confidence in our products and in Lenovo.”

“The BSIMM community itself is also a fantastic resource with members generously sharing experiences and lessons learned. We’re all on a similar journey, and those firms just starting their activities can learn so much from those that started earlier.”

— BILL JAEGER, EXECUTIVE DIRECTOR OF LENOVO’S ISG PRODUCT SECURITY OFFICE.

The BSIMM Experience

- Lenovo’s latest BSIMM assessment, conducted in 2021, found the company’s product security program to be in the top percentile of software security initiatives, more mature than typical for a relatively young security program.
- “Our experience with BSIMM has been highly positive,” Jaeger notes. “We joined the BSIMM community in 2015 and have found significant value in BSIMM’s annually refreshed real-world activity observations. They help us plan and measure our own security program and gain a sense of the practice areas that are most important to our customers in each industry vertical.”
- “The BSIMM community itself is also a fantastic resource with members generously sharing experiences and lessons learned. We’re all on a similar journey, and those firms just starting their activities can learn so much from those that started earlier.”

CASE STUDY: LEADING NORTH AMERICAN FINANCIAL INSTITUTION

A leading North American financial institution offers a full range of advice, solutions, and services through its digital banking network and locations around the world across personal, business banking, and commercial banking.

“We’ve been building and maturing our software security initiative since 2013 and continue to introduce new security capabilities as our business needs demand,” says their senior manager of application security. “Our current programs include building and publishing application security standards and guidelines; software security training and awareness; applying security testing during development as well as deploying runtime vulnerability detection and protection. We’re also in the process of revamping our security champions program and have a DevSecOps pilot that we plan to roll out enterprise wide.”

“We became a member of the BSIMM community in 2015,” he continues. “BSIMM offers one of the best security reference and guidance models available. We highly value the BSIMM framework and description of security practices. We’ve found that the BSIMM

assessment and comparison against our peers is extremely useful information for setting the direction and scope of our security efforts.”

A BSIMM assessment provides an objective, data-driven evaluation for organizations seeking to improve their security postures and can be used for decisions about allocation of resources, time, budget, and priorities. “The assessment helps us understand how our peers are doing, and the most important and least important security practices are in our industry,” says the senior manager of application security. “It’s a great reference source to help us build out our security roadmaps.”

“BSIMM offers one of the best security reference and guidance models available industrywide.”

—SENIOR MANAGER OF APPLICATION SECURITY

CASE STUDY: CRED

“We wanted to do an industry benchmark of the security process established within CRED and understand where we stand with respect to other organizations in the world.”

An exclusive community where members are rewarded for good financial behavior, CRED was born out of a need to bring back the focus on trust, the idea being to create a community centered around this virtue. CRED as an institution has a solid reputation of providing a wide variety of product offerings to its members—from lifestyle services to personal finance.

“Security has been ingrained into our culture since inception,” says Himanshu Kumar Das, CISO, CRED. “A ‘security-first’ company,

CRED has its software security initiatives in place from day 1 and during our 3+ years of existence, we have established maturity in multiple key disciplines of security.”

“We have around 350+ internal micro services which are updated multiple times a day with changes deployed in several iterations,” says Anirudh Anand, Team Lead, Product Security. “Thorough security review of these changes is performed on a regular basis during release cycles. CRED Android/iOS mobile applications are fully reviewed before being shipped fortnightly. We also perform weekly, quarterly, and annual vulnerability assessment and penetration testing (VAPT) as part of the vulnerability management process.”

THE BSIMM ONLINE COMMUNITY

The BSIMM is not just a report of the state of the industry—it's bigger than that. The BSIMM online community is where software security leaders come to learn, share ideas, and get customized information to reduce their software risk.

The online community is a unique, members-only forum that helps members address software security challenges in today's complex business environments.

Member benefits in the BSIMM community include:

- Discussion forum with peers
- Free e-learning courses
- Original content (blogs, profiles)
- Two annual conferences
- Conference archives
- Webinars and podcasts
- Industry reports and more

From content authored by industry-leaders to hands-on interactions with fellow BSIMM members, it's a powerful resource for collaborative problem solving, thought leadership, and access to valuable resources not available anywhere else.

ACKNOWLEDGEMENTS


Our thanks to the 130 executives, including those who wish to remain anonymous, from the SSIs we studied to create BSIMM13.

Our thanks also to the nearly 140 individuals who helped gather the data for the BSIMM data pool over time.

In particular, we thank Tony Blakemore, Adam Brown, Matthew Chartrand, Eli Erlikhman, Jacob Ewers, Stephen Gardner, iMan Louis, Sammy Miguez, Alistair Nash, Kevin Nassery, Donald Pollicino, Brendan Sheairs, Denis Sheridan, and Li Zhao.

BSIMM13 was authored by Jamie Boote, Eli Erlikhman, Stephen Gardner, and Sammy Miguez. In addition, we give a special thank you to Kathy Clark-Fisher and Ryan Francis, whose behind-the-scenes work keeps the BSIMM science project, conferences, and community on track.

AARP	F-Secure	PayPal
Adobe	Genetec	Pegasystems
Aetna	HCA Healthcare	Principal Financial
Ally Bank	Honeywell CE	Realtek
Axway	HSBC	Reckitt
Bank of America	Imperva	SambaSafety
Bell Network	Inspur Software	ServiceNow
CIBC	Intralinks	Signify
Cisco	iPipeline	SonicWall
Citi	Johnson & Johnson	Synchrony Financial
Diebold Nixdorf	Landis+Gyr	TD Ameritrade
Depository Trust & Clearing Corporation	Lenovo	Teradata
Egis	MassMutual	Trainline
Eli Lilly and Company	MediaTek	Trane
eMoney Advisor	Medtronic	U.S. Bank
EQBank	Navient	Veritas
Equifax	Navy Federal Credit Union	Verizon Media
Fidelity	NEC	Vivo
Finastra	NetApp	World Wide Technology
Freddie Mac	Oppo	ZoomInfo



PART 4:
QUICK GUIDE
TO SSI MATURITY

QUICK GUIDE TO SSI MATURITY

Eleven questions can help clarify where your SSI is today. Combined with a detailed software security scorecard (see below on how to measure your own program) and knowledge about roles and responsibilities, you can use this information to plan strategic changes for ongoing success.

SSI maturity is a complex thing. Each organization will apply different values to efforts and progress in people, process, technology, and culture. They will also evolve differently in their vision for success as well as how they spend resources, grow the program, and manage risk. This section provides an approach to organizing, growing, and maturing an SSI that works for everyone. Refer to Appendix B for more details.

QUICK BASELINE FOR SSI LEADERS

All program leaders require a detailed understanding of their efforts and whether those efforts align with business objectives. A good start here is to understand whether organizational SSI efforts align well with changes in the software security landscape driven by global events, digital transformation, and engineering evolution, as well as with how software is made today. Use your answers to the questions below to determine whether it's time to invest in new growth. If you don't know all these answers, use the list to gather information from every SSI stakeholder responsible for aspects of software security risk management.

Is Your SSI Keeping Pace with Change in Your Software Portfolio?

- Do you maintain a current view of all your software assets, including internal code, third-party code, open source, automation scripts, infrastructure-as-code, and other software assets?
- Are you using SBOMs that detail all components in the SSI's software portfolio in your risk management processes?
- Do you have a near-real-time view of the software deployments in your operations environments, along with a view into their aggregate attack surface and aggregate risk?

Are You Creating the DevSecOps Culture You Need?

- Are you building bridges between the various software security stakeholders in your organization—governance, technical, audit, vendor management, cloud, and so on—to align culture, approach, technology stacks, and testing strategies?
- Have you scaled your satellite group program across your software portfolio, including skills specific to automation, technology stacks, application architectures, and other important needs?
- Are you delivering important security policy, standards, and guidelines as code that runs in engineering and operations toolchains?

Are You Shifting Security Efforts Everywhere in the Engineering Lifecycle?

- Are you automating security decisions to remove time-consuming manual review and moving toward an auditable, governance-as-code secure SDLC?
- Are you following a shift everywhere strategy to move from large, time-consuming security tests to smaller, faster, timely, pipeline-driven security tests conducted to improve engineering team performance?
- Are you looking in the source code, builds, and operational software for malicious code that might have been introduced into your critical software, whether that software is developed internally or externally?

How Does Your SSI Measure Up?

- Do you routinely use telemetry from security testing, operations events, risk management processes, event postmortems, and other efforts to drive process improvements in your secure SDLC or governance improvements in your policies and standards?
- Does your SSI strategy account for the impact on software security caused by adjacent disciplines that require their own security efforts, such as cloud, container, orchestration, source content management, development pipeline, shared responsibility models, and so on?

Most organizations have already covered the basics of software security policy, testing, and outreach. It takes a concerted effort to scale an SSI to address changes in portfolio size, technology, infrastructure, regulation, laws, attackers, attacks, and more. Internal review and reflection on efforts versus needs is always a good way to move forward.

USING A BSIMM SCORECARD TO MAKE PROGRESS

A BSIMM scorecard is a management tool that allows your SSI and SSG leadership to:

- Assess your level of maturity so you can evolve your software security journey in stages, first building a strong emerging foundation, then maturing the more complex activities over time.
- Communicate your software security posture to customers, partners, executives, and regulators. A scorecard helps everyone understand where you are in your journey and where you want to go when you're explaining your strategic plan and budgets.
- See actual measurement data from the field. This helps in building a long-term plan for an SSI and in tracking progress against that plan.

In addition to being a lens on the state of software security, the BSIMM serves as a measuring stick to determine where your SSI currently stands relative to the community, whether as a whole or for specific verticals. For example, a direct comparison of your efforts across all 125 activities to the BSIMM13 scorecard for the entire community (see Appendix D) is probably the best first step. Follow the steps below to use the BSIMM to create your own SSI scorecard (see Figure 3 for an example).

Understand Your Organizational Mandate

- Decide what the SSI intends to accomplish. Who are the executive sponsors, and what resources are they expected to provide? From a RACI perspective, who are the responsible and accountable stakeholders? What metrics must be provided to executive management to demonstrate acceptable progress?
- Set the proper scope for the SSI. At a high level, describe the applicable software portfolio and the associated software ownership (e.g., risk managers). Ensure that you include all applications and related software that's in the SSG's remit.

Build the Scorecard

- Make a list of stakeholders to interview. No single person knows everything about a modern SSI, so ensure that you have broad coverage across the SSG, satellite (your champions), engineering, QA, operations, and security testing. As needed, extend the stakeholder list to include teams from reliability, cloud, privacy, training, infrastructure, and others whose efforts have a direct impact on software security.
- Understand the BSIMM. Review the BSIMM activities and gain an understanding of the practices, the individual activities, and the themes that run through them. For example, the activities for software security testing appear across multiple BSIMM practices.
- Interview everyone and consolidate the results. Keep interviews brief and focused but ensure that you get the data and artifacts that demonstrate the organization is sufficiently—in both depth and breadth—performing each activity before you award credit.

- Create your scorecard. Use a binary one or zero, a scale of low, medium, and high, or even a graduated scale such as a percentage to combine aspects of depth, breadth, and maturity.

Make a Strategic Plan and Execute

- Compare your scorecard to your stakeholders' realistic expectations. Prioritize effort on the important gaps as well as those gaps with a long lead time. See Appendix B for more details on how to build an execution plan. Mark your calendar to revisit the scorecard in 12 to 18 months, document your progress, and create a new scorecard.
- Define and use metrics to gauge progress. Every program needs a barometer for success, and each organization finds different things to be the best indicators for them. Whether described as metrics, KPIs, KRIs, SLOs, or something else, use what works best for you.

For most organizations, a single aggregated scorecard covering the entire SSI will suffice to inform future planning. In some cases, however, it will be beneficial to create individual scorecards for the SSG and for business units or application teams that have varying software security approaches or maturity levels.

Figure 3 depicts an example firm that performs 41 BSIMM13 activities (noted as 1s in its EXAMPLEFIRM scorecard columns, e.g., SM1.1), including nine activities that are the most common in their respective practices (orange, e.g., CP1.2). Note the firm does not perform the most observed activities in the other three practices (gray boxes, e.g., SM1.4) and should take some time to determine whether these are necessary or useful to its overall SSI. The BSIMM13 FIRMS columns show the number of observations (currently out of 130) for each activity, allowing the firm to understand the activity's general popularity within the current community. If you want to evaluate your scorecard against a particular vertical, refer to Appendix E.

Once you have determined where you stand with activities compared to your expectations, you can devise a plan for improvement. Organizations almost always choose some hybrid of expanding their SSI with new activities and scaling some existing activities across more of the software portfolio and stakeholder teams.

Note that there's no inherent reason to adopt all activities in each practice. Prioritize the ones that make sense for your organization today and set aside those that don't—but revisit those choices periodically. Once they've adopted an activity set, most organizations strategically work on the depth, breadth, and cost-effectiveness (e.g., via automation) of each activity in accordance with their view of the risk management efforts required in their environments for their business objectives.

To help refine the current and future activity prioritization for your SSI, you can go beyond the AllFirms data in Appendix D, Figure 17 and analyze how SSIs evolve with remeasurements (Appendix F) and with age (Appendix H). You can also examine what's different about your vertical or verticals (Appendix E) and understand the impact of a champions program (Appendix G) on SSIs.

GOVERNANCE			INTELLIGENCE			SSDL TOUCHPOINTS			DEPLOYMENT		
ACTIVITY	BSIMM13 FIRMS (OUT OF 130)	EXAMPLE FIRM	ACTIVITY	BSIMM13 FIRMS (OUT OF 130)	EXAMPLE FIRM	ACTIVITY	BSIMM13 FIRMS (OUT OF 130)	EXAMPLE FIRM	ACTIVITY	BSIMM13 FIRMS (OUT OF 130)	EXAMPLE FIRM
STRATEGY & METRICS			ATTACK MODELS			ARCHITECTURE ANALYSIS			PENETRATION TESTING		
[SM1.1]	98	1	[AM1.2]	80		[AA1.1]	113	1	[PT1.1]	114	
[SM1.3]	82		[AM1.3]	42		[AA1.2]	53	1	[PT1.2]	102	1
[SM1.4]	117		[AM1.5]	76	1	[AA1.4]	69		[PT1.3]	88	1
[SM2.1]	73		[AM2.1]	16		[AA2.1]	31		[PT2.2]	38	
[SM2.2]	63		[AM2.2]	11	1	[AA2.2]	32	1	[PT2.3]	45	
[SM2.3]	69		[AM2.5]	16	1	[AA2.4]	38	1	[PT3.1]	26	1
[SM2.6]	71		[AM2.6]	16		[AA3.1]	20		[PT3.2]	15	
[SM2.7]	64	1	[AM2.7]	14		[AA3.2]	4				
[SM3.1]	27		[AM3.1]	9		[AA3.3]	15				
[SM3.2]	18		[AM3.2]	5							
[SM3.3]	26		[AM3.3]	11							
[SM3.4]	5										
[SM3.5]	0										
COMPLIANCE & POLICY			SECURITY FEATURES & DESIGN			CODE REVIEW			SOFTWARE ENVIRONMENT		
[CP1.1]	101	1	[SFD1.1]	104	1	[CR1.2]	83	1	[SE1.1]	87	
[CP1.2]	115	1	[SFD1.2]	90	1	[CR1.4]	107	1	[SE1.2]	115	1
[CP1.3]	98	1	[SFD2.1]	39		[CR1.5]	62		[SE1.3]	79	1
[CP2.1]	58		[SFD2.2]	64		[CR1.7]	54		[SE2.2]	57	1
[CP2.2]	59		[SFD3.1]	17		[CR2.6]	28	1	[SE2.4]	39	
[CP2.3]	73		[SFD3.2]	18		[CR2.7]	20		[SE2.5]	52	1
[CP2.4]	62		[SFD3.3]	7		[CR2.8]	34	1	[SE2.7]	42	1
[CP2.5]	82	1				[CR3.2]	14		[SE3.2]	19	
[CP3.1]	30					[CR3.3]	8		[SE3.3]	11	
[CP3.2]	28					[CR3.4]	2		[SE3.6]	18	
[CP3.3]	11					[CR3.5]	3		[SE3.8]	0	
TRAINING			STANDARDS & REQUIREMENTS			SECURITY TESTING			CONFIG. MGMT. & VULN. MGMT.		
[T1.1]	71	1	[SR1.1]	96	1	[ST1.1]	108	1	[CMVM1.1]	114	1
[T1.7]	58	1	[SR1.2]	101		[ST1.3]	97	1	[CMVM1.2]	100	
[T1.8]	53		[SR1.3]	103	1	[ST1.4]	56		[CMVM2.1]	95	1
[T2.5]	38		[SR2.2]	80	1	[ST2.4]	25		[CMVM2.2]	98	
[T2.8]	28	1	[SR2.4]	92		[ST2.5]	31		[CMVM2.3]	62	
[T2.9]	33	1	[SR2.5]	63	1	[ST2.6]	21		[CMVM3.1]	11	
[T2.10]	28		[SR2.7]	53		[ST3.3]	12		[CMVM3.2]	19	
[T2.11]	27		[SR3.2]	19		[ST3.4]	4		[CMVM3.3]	18	
[T3.1]	9		[SR3.3]	17		[ST3.5]	4		[CMVM3.4]	26	1
[T3.2]	16		[SR3.4]	19		[ST3.6]	3		[CMVM3.5]	13	1
[T3.5]	22								[CMVM3.6]	3	
[T3.6]	7								[CMVM3.7]	20	
									[CMVM3.8]	0	

FIGURE 3. BSIMM13 EXAMPLEFIRM SCORECARD. A scorecard helps everyone understand the software security efforts that are currently underway. It also helps organizations make comparisons to the community and serves as a guide on where to focus next.

ROLES IN A SOFTWARE SECURITY INITIATIVE

Determining the right activities to focus on and clarifying who is responsible for their implementation are important parts of making any SSI work. That means putting people in leadership roles and giving them clear responsibilities and objectives.


From our work with 254 BSIMM participants since 2008, we've observed the following software security roles and responsibilities being important across a wide variety of organizations of different sizes, in different verticals, and with both large and small remits (e.g., application portfolio size):

- **Executive leadership.** As an SSI takes shape and requires dedicated resources, it also requires an executive sponsor to own the initiative, define objectives, provide budget and people, and ensure progress. Executive leadership must help translate business objectives into security objectives in one direction and help translate security data into risk data in the other.
- **SSG leader.** An SSI looking to grow needs an SSG dedicated to scaling the program across the organization. The SSG leader and their team must execute on the security objectives across an array of stakeholders, including development, QA, and operations. This will require starting and maturing software capabilities such as defect discovery and management, software supply chain security, training, and telemetry and metrics.
- **Satellite (security champions).** Very few SSGs can become large enough to do their business-as-usual tasks and also be responsive to all stakeholders all the time. A security champions group is an effective way to scale SSG reach by embedding trained experts in stakeholder business processes. Security champions take on tasks such as running security tools and doing testing results triage, on-demand training, research on complicated security issues, and ensuring that software security checkpoints are passed successfully.

- **Architects and developers.** Even the best policy and process can't guarantee secure software. People designing and coding software must practice good security engineering, follow designated procedures for responding to discovered security issues, and collaborate actively with other stakeholders. Architects and developers are often a source of innovation in security integration and as-code improvements, so it's important to share these ideas broadly.
- **QA teams.** Code functionality is obviously critical to organizational success, but getting QA teams to include security tests in their automated suites provides an easy way to expand the search for security defects. QA teams can also be a source of innovation for automating security tests in preproduction environments.
- **Operations and administration.** Even the most secure code can be undermined by poor host, network, cloud, or other configurations and administration. Operations teams have an opportunity to ensure that configurations, administration, access controls, logging, monitoring, and as-code efforts support software security objectives.
- **Data privacy.** Specialists can help ensure that regulations, laws, contracts, and client expectations are translated into software requirements.

Refer to Appendix A for more details on roles and responsibilities.

Determining the right activities to focus on and clarifying who is responsible for their implementation are important parts of making any SSI work.



PART 5:
THE BSIMM
FRAMEWORK

THE BSIMM FRAMEWORK

- Most of the BSIMM will likely fit perfectly for your SSI, but some parts might feel a little less applicable.
- Understanding the model allows you to both learn from others and ensure that your program is right for your organization.

We built the first version of the BSIMM nearly 14 years ago (late 2008) as follows:

- We relied on our own knowledge of software security practices to create the initial SSF.
- We conducted a series of in-person interviews with nine executives in charge of SSIs. From these interviews, we identified a set of 110 software security activities that we organized according to the SSF.
- We then created scorecards for each of the nine initiatives that showed which of the activities each initiative carried out. To validate our work, we asked each participating firm to review the SSF, practices, activities, and the scorecard we created for their initiative, then we made necessary adjustments based on their feedback.

Today, we continue to do BSIMM assessments with in-person interviews whenever possible, which we've done with a total of 254 firms so far. In addition, we've conducted assessments for 14 organizations that have rejoined the community after aging out. In 43 cases, we assessed both the SSG and one or more business units as part of creating an aggregated SSI view for a firm. We evolve the model by digging for new kinds of efforts during assessments—both as new participants join and as current participants are

remeasured—and then adding new activities when warranted, and we've added 16 since 2008. We also adjust the positioning of activities in the model practices according to their observation rates.

CORE KNOWLEDGE

The BSIMM core knowledge encompasses the activities we have directly observed in the BSIMM community—the group of firms that participate in using the BSIMM as part of their SSI management. We organize that core knowledge into an SSF, represented in Figure 4. The SSF is organized into four domains—Governance, Intelligence, SSDL Touchpoints, and Deployment—with those domains containing the 125 BSIMM13 activities.

From an executive perspective, you can view BSIMM activities as controls implemented in a software security risk management framework. The implemented activities might function as preventive, detective, corrective, or compensating controls in your SSI. Positioning the activities as controls allows for easier understanding of the BSIMM's value by governance, risk, compliance, legal, audit, and other risk management groups.

We divide activities into levels per practice based on the frequency with which they're observed in the community. We do this to help organizations quickly understand whether the activity they're contemplating is common or uncommon across other organizations. Level 1 activities (often straightforward and universally applicable) are those that are most observed across the community of 130 firms, level 2 (often more difficult to implement and requiring more coordination) are less frequently observed, and level 3 activities (usually more difficult to implement and not always applicable) are more rarely observed. Note that new activities are added at level 3 because we don't yet know how common they are, so they start with zero observations.

DOMAINS			
GOVERNANCE	INTELLIGENCE	SSDL TOUCHPOINTS	DEPLOYMENT
Practices that help organize, manage, and measure a software security initiative. Staff development is also a central governance practice.	Practices that result in collections of corporate knowledge used in carrying out software security activities throughout the organization. Collections include both proactive security guidance and organizational threat modeling.	Practices associated with analysis and assurance of particular software development artifacts and processes. All software security methodologies include these practices.	Practices that interface with traditional network security and software maintenance organizations. Software configuration, maintenance, and other environment issues have direct impact on software security.
PRACTICES			
GOVERNANCE	INTELLIGENCE	SSDL TOUCHPOINTS	DEPLOYMENT
1. Strategy & Metrics (SM) 2. Compliance & Policy (CP) 3. Training (T)	4. Attack Models (AM) 5. Security Features & Design (SFD) 6. Standards & Requirements (SR)	7. Architecture Analysis (AA) 8. Code Review (CR) 9. Security Testing (ST)	10. Penetration Testing (PT) 11. Software Environment (SE) 12. Configuration Management & Vulnerability Management (CMVM)

FIGURE 4. THE SOFTWARE SECURITY FRAMEWORK. Twelve practices align with the four high-level domains and contain the 125 BSIMM13 activities.

UNDERSTANDING THE MODEL

A domain, such as Governance, contains practices, such as Strategy & Metrics, each of which contains activities that each have a detailed description. Creating a scorecard (e.g., activity SM1.1 was observed and is marked with a "1") informs decisions about strategic change.

GOVERNANCE

1. **Strategy & Metrics (SM)**
2. Compliance & Policy (CP)
3. Training (T)

GOVERNANCE			
STRATEGY & METRICS			
[SM1.1]	Publish process and evolve as necessary.	[SM2.7]	Create evangelism role and perform internal marketing.
[SM1.3]	Educate executives on software security.	[SM3.1]	Use a software asset tracking application with portfolio view.
[SM1.4]	Implement security checkpoints and associated governance.	[SM3.2]	Make SSI efforts part of external marketing.
[SM2.1]	Publish data about software security internally and use it to drive change.	[SM3.3]	Identify metrics and use them to drive resourcing.
[SM2.2]	Enforce security checkpoints and track exceptions.	[SM3.4]	Integrate software-defined lifecycle governance.
[SM2.3]	Create or grow a satellite (security champions).	[SM3.5]	Integrate software supply chain risk management.
[SM2.6]	Require security sign-off prior to software release.		

[SM2.2: 63]

Enforce security checkpoints and track exceptions.

Enforce security release conditions at each checkpoint (gate, guardrail, milestone, etc.) for every project, so that each project must either meet an established measure or follow a defined process for obtaining an exception to move forward. Use internal policies and standards, regulations, contractual agreements, and other obligations to define release conditions, then track all exceptions. Verifying conditions yields data that informs the KRIs and any other metrics used to govern the process. Automatically giving software a passing grade or granting exceptions without due consideration defeats the purpose of verifying conditions. Even seemingly innocuous software projects (e.g., small code changes, infrastructure access control changes, deployment blueprints) must successfully satisfy the prescribed security conditions as they progress through the software lifecycle. Similarly, APIs, frameworks, libraries, bespoke code, microservices, container configurations, and so on are all software that must satisfy security release conditions. It's possible, and often very useful, to have verified the conditions both before and after the development process itself. In modern development environments, the verification process will increasingly become automated.

GOVERNANCE

ACTIVITY	BSIMM13 FIRMS (OUT OF 130)	EXAMPLE FIRM
STRATEGY & METRICS		
[SM1.1]	98	1
[SM1.3]	82	
[SM1.4]	117	
[SM2.1]	73	
[SM2.2]	63	
[SM2.3]	69	
[SM2.6]	71	
[SM2.7]	64	1
[SM3.1]	27	
[SM3.2]	18	
[SM3.3]	26	
[SM3.4]	5	
[SM3.5]	0	



PART 6:
THE BSIMM
ACTIVITIES

THE BSIMM ACTIVITIES

- The BSIMM activities are the individual controls used to construct or improve an SSI. They range through people, process, technology, and culture. You can use this information to choose which controls to apply within your initiative, then align your implementation strategy and metrics with your desired outcomes.

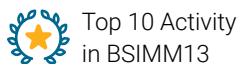
The BSIMM framework comprises four domains—Governance, Intelligence, SSDL Touchpoints, Deployment—and those domains contain 12 practices, such as Strategy & Metrics, Attack Models, and Code Review, which each contain activities. These activities are the BSIMM building blocks, the smallest unit of software security granularity implemented to build SSIs. Rather than prescriptively dictating a set of best practices, the BSIMM descriptively observes, quantifies, and documents the actual activities carried out by various kinds of SSIs across diverse organizations.

ACTIVITIES IN THE BSIMM

The BSIMM is a data-driven model that evolves over time. Over the years, we have added, deleted, and adjusted the levels of various activities based on the data observed throughout the BSIMM's evolution. When considering whether to add a new activity, we analyze whether the effort we're observing is truly new to the model or simply a variation on an existing activity. Similarly, for deciding whether to move an activity between levels within a practice, we use the results of an intra-level standard deviation analysis and the trend in observation counts.

Each activity has a unique label and name—for example, activity SM1.4 is in the Strategy & Metrics practice and is named *Implement security checkpoints and associated governance*. To preserve backward compatibility, we make all changes by adding new activity labels to the model, even when an activity has simply changed levels within a practice (as an example, we would add a new CR#. # label for both new and moved activities in the Code Review practice).

BSIMM activity levels distinguish the frequency with which activities are observed in the participating organizations. As seen in Part 5, frequently observed activities are designated level 1, with less frequent and infrequently observed activities designated as levels 2 and 3, respectively. Using SM1.4 as an example again, we see that it is a frequently observed activity in the Strategy & Metrics practice. Note that the new activities we add to the model start with zero observations and are therefore always added at level 3.



Top 10 Activity
in BSIMM13



New Activity
in BSIMM13

GOVERNANCE

Governance: Strategy & Metrics (SM)

The Strategy & Metrics practice encompasses planning, assigning roles and responsibilities, identifying software security goals, determining budgets, and identifying metrics and software release conditions.

[SM1.1: 98] Publish process and evolve as necessary.

The process for addressing software security is defined, published internally, and broadcast to all stakeholders so that everyone knows the plan. Goals, roles, responsibilities, and activities are explicitly defined. Most organizations examine existing methodologies, such as the NIST SSDF, Microsoft SDL, or Synopsys Touchpoints, then tailor them to meet their needs. Security activities will be adapted to software lifecycle processes (e.g., waterfall, Agile, CI/CD, DevOps), so activities will evolve with both the organization and the security landscape. The process doesn't need to be publicly promoted outside the firm to have the desired impact (see [SM3.2]). In addition to publishing the written process, some firms also automate parts (e.g., a testing strategy) as governance-as-code (see [SM3.4]).

[SM1.3: 82] Educate executives on software security.

Executives are regularly shown the ways malicious actors attack software and the negative business impacts those attacks can have on the organization. Go beyond reporting of open and closed defects to educate executives on the business risks, including risks of adopting emerging engineering technologies and methodologies without security oversight. Demonstrate a worst-case scenario in a controlled environment with the permission of all involved (e.g., by showing attacks and their business impact). Presentation to the Board can help garner resources for new or ongoing SSI efforts. Demonstrating the need for new skill-building training in evolving areas, such as DevOps groups using cloud-native technologies, can help convince leadership to accept SSG recommendations when they might otherwise be ignored in favor of faster release dates or other priorities. Bring in an outside expert when necessary to bolster executive attention.



[SM1.4: 117] Implement security checkpoints and associated governance.

The software security process includes checkpoints (such as gates, release conditions, guardrails, milestones, etc.) at one or more points in a software lifecycle. The first two steps toward establishing security-specific checkpoint conditions are to identify process locations that are compatible with existing development practices and to then begin gathering the information necessary to make a go/no-go decision, such as risk-ranking thresholds or defect data. Importantly, the conditions need not be enforced at this stage—for example, the SSG can collect security testing results for each project prior to release, then provide their informed opinion on what constitutes sufficient testing or acceptable test results without trying to stop a project from moving forward (see [SM2.2]). Shorter release cycles might require creative approaches to collecting the right evidence and rely heavily on automation. Socializing the conditions and then enforcing them once most project teams already know how to succeed is a gradual approach that motivates good behavior without introducing unnecessary friction.

[SM2.1: 73] Publish data about software security internally and use it to drive change.

To facilitate improvement, data is published internally about the state of software security within the organization. Produce security or development dashboards with metrics for executives and software development management. Dashboards can be part of pipeline toolchains to enable developer self-improvement. Sometimes, this published data won't be shared with everyone in the firm but only

with the stakeholders who are tasked to drive change. In other cases, open book management and data published to all stakeholders helps everyone know what's going on. If the organization's culture promotes internal competition between groups, use this information to add a security dimension. Integrate automated security telemetry to gather measurements quickly and accurately to increase timeliness of security data in areas such as speed (e.g., time to fix) and quality (e.g., defect density). Publishing data about new technologies (e.g., security and risk in cloud-native architectures) is important for identifying needed improvements.

[SM2.2: 63] Enforce security checkpoints and track exceptions.

Enforce security release conditions at each checkpoint (gate, guardrail, milestone, etc.) for every project, so that each project must either meet an established measure or follow a defined process for obtaining an exception to move forward. Use internal policies and standards, regulations, contractual agreements, and other obligations to define release conditions, then track all exceptions. Verifying conditions yields data that informs the KRIs and any other metrics used to govern the process. Automatically giving software a passing grade or granting exceptions without due consideration defeats the purpose of verifying conditions. Even seemingly innocuous software projects (e.g., small code changes, infrastructure access control changes, deployment blueprints) must successfully satisfy the prescribed security conditions as they progress through the software lifecycle. Similarly, APIs, frameworks, libraries, bespoke code, microservices, container configurations, and so on are all software that must satisfy security release conditions. It's possible, and often very useful, to have verified the conditions both before and after the development process itself. In modern development environments, the verification process will increasingly become automated (see [SM3.4]).

[SM2.3: 69] Create or grow a satellite (security champions).

Form a collection of people scattered across the organization—a satellite—who show an above-average level of security interest or skill and who contribute software security expertise to development, QA, and operations teams. Forming this social network of advocates, sometimes referred to as champions, is a good step toward scaling security into software engineering. One way to build the initial group is to track the people who stand out during introductory training courses (see [T3.6]). Another way is to ask for volunteers. In a more top-down approach, initial satellite membership is assigned to ensure good coverage of development groups, but ongoing membership is based on actual performance. The satellite can act as a sounding board for new projects and, in new or fast-moving technology areas, help combine software security skills with domain knowledge that might be under-represented in the SSG or engineering teams. Agile coaches, scrum masters, and DevOps engineers can make particularly useful satellite members, especially for detecting and removing process friction. In some environments, satellite-led efforts are being delivered via automation.

[SM2.6: 71] Require security sign-off prior to software release.

The organization has an initiative-wide process for documenting accountability and accepting security risk by having a risk owner sign off on the state of all software prior to release based on SSG-approved criteria. The sign-off policy might, for example, also require

the accountable person to acknowledge critical vulnerabilities that have not been mitigated or SSDL steps that have been skipped. Informal or uninformed risk acceptance alone isn't a security sign-off because the act of accepting risk is more effective when it's formalized (e.g., with a signature, a form submission, or something similar) and captured for future reference. Similarly, simply stating that certain projects don't need sign-off at all won't achieve the desired risk management results. In some cases, however, the risk owner can provide the sign-off on a particular set of software project acceptance criteria, which are then implemented in automation to provide governance-as-code (see [SM3.4]), but there must be an ongoing verification that the criteria remain accurate, and the automation is working.

[SM2.7: 64] Create evangelism role and perform internal marketing.

Build support for software security throughout the organization via ongoing evangelism. This internal marketing function, often performed by a variety of stakeholder roles, keeps executives and others up to date on the magnitude of the software security problem and the elements of its solution. A scrum master familiar with security, for example, could help teams adopt better software security practices as they transform to Agile and DevOps methods. Similarly, a cloud expert could demonstrate the changes needed in security architecture and testing for serverless applications. Evangelists can increase understanding and build credibility by giving talks to internal groups (including executives), publishing roadmaps, authoring technical papers for internal consumption, or creating a collection of papers, books, and other resources on an internal website (see [SR1.2]) and promoting its use. In turn, organizational feedback becomes a useful source of improvement ideas.

[SM3.1: 27] Use a software asset tracking application with portfolio view.

The SSG uses centralized tracking automation to chart the progress of every piece of software and deployable artifact from creation to decommissioning, regardless of development methodology. The automation records the security activities scheduled, in progress, and completed, incorporating results from SSDL activities even when they happen in a tight loop or during deployment. The combined inventory and security posture view enables timely decision-making. The SSG uses the automation to generate portfolio reports for multiple metrics and, in many cases, publishes this data at least among executives. As an initiative matures and activities become more distributed, the SSG uses the centralized reporting system to keep track of all the moving parts.

[SM3.2: 18] Make SSI efforts part of external marketing.

To build external awareness, the SSG helps market the SSI beyond internal teams. In this way, software security can grow its risk reduction exercises into a competitive advantage or market differentiator. The SSG might publish papers or books about its software security capabilities or have a public blog. It might provide details at external conferences or trade shows. In some cases, a complete SSDL methodology can be published and promoted outside the firm, and governance-as-code concepts can make interesting case studies. Regardless of method, the process of sharing details externally and inviting critique is used to bring new perspectives into the firm.

[SM3.3: 26] Identify metrics and use them to drive resourcing.

The SSG and its management identify metrics that define and measure SSI progress in quantitative terms. These metrics are reviewed on a regular basis and drive the initiative's budgeting and resource allocations, so simple counts and out-of-context measurements won't suffice here. On the technical side, one such metric could be defect density, a reduction of which could be used to show a decreasing cost of remediation over time, assuming, of course, that testing depth has kept pace with software changes. Data for metrics is best collected early and often using event-driven processes with telemetry rather than calendar-driven data collection. The key is to tie security results to business objectives in a clear and obvious fashion to justify resourcing. Because the concept of security is already tenuous to many businesspeople, make the tie-in explicit.

[SM3.4: 5] Integrate software-defined lifecycle governance.

Organizations begin replacing traditional document-, presentation-, and spreadsheet-based lifecycle management with software-based delivery platforms. For some software lifecycle phases, humans are no longer the primary drivers of progression from one phase to the next. Instead, organizations rely on automation to drive the management and delivery process with software such as Spinnaker or GitHub, and humans participate asynchronously (and often optionally). Automation often extends beyond the scope of CI/CD to include functional and nonfunctional aspects of delivery, such as health checks, cut-over on failure, rollback to known-good state, defect discovery and management, compliance verification, and a way to ensure adherence to policies and standards. Some organizations are also evolving their lifecycle management approach by integrating their compliance and defect discovery data, perhaps augmented by intelligence feeds and other external data, to begin moving from a series of point-in-time go/no-go decisions (e.g., release conditions) to a future state of continuous accumulation of assurance data (see [CMVM3.6]).



[SM3.5: 0] Integrate software supply chain risk management.

Organizational risk management processes ensure that important software created by and entering the organization is managed through governance-driven access and usage controls, maintenance standards, and captured provenance data. Apply these processes to external (see [SR2.7]), bespoke, and internally developed software, helping to ensure that deployed code has the expected components (see [SE3.8]). The lifecycle management for all software, from creation or importation through secure deployment, ensures that all access, usage, and modifications are done in accordance with policy. This assurance is easier to implement at scale using automation in software lifecycle processes (see [SM3.4]).

Governance: Compliance & Policy (CP)

The Compliance & Policy practice is focused on identifying controls for compliance regimens such as PCI DSS and GDPR, developing contractual controls such as SLAs to help manage COTS software risk, setting organizational software security policy, and auditing against that policy.

[CP1.1: 101] Unify regulatory pressures.

Have a central team to understand the constraints imposed on software security by regulatory or compliance drivers applicable to the organization and its customers. The team creates or collaborates on a unified approach that removes redundancy and conflicts from overlapping compliance requirements, such as from PCI security standards; GLBA, SOX, and HIPAA in the US; or GDPR in the EU. A formal approach will map applicable portions of regulations to controls (see [CP2.3]) applied to software to explain how the organization complies. Existing business processes run by legal, product management, or other risk and compliance groups outside the SSG could serve as the regulatory focal point, with the SSG providing software security knowledge. A unified set of software security guidance for meeting regulatory pressures ensures that compliance work is completed as efficiently as possible.



[CP1.2: 115] Identify privacy obligations.

The SSG identifies privacy obligations stemming from regulation and customer expectations, then translates these obligations into both software requirements and privacy best practices. The way software handles PII might be explicitly regulated, but even if it isn't, privacy is an important topic. For example, if the organization processes credit card transactions, the SSG will help in identifying the constraints that the PCI DSS places on the handling of cardholder data and will inform all stakeholders (see [SR1.3]). Note that outsourcing to hosted environments (e.g., the cloud) doesn't relax privacy obligations and can even increase the difficulty of recognizing and meeting all associated needs. Also note that firms creating software products that process PII when deployed in customer environments might meet this need by providing privacy controls and guidance for their customers. Evolving consumer privacy expectations, the proliferation of "software is in everything," and data scraping and correlation (e.g., social media) add additional expectations for PII protection.

[CP1.3: 98] Create policy.

The SSG guides the organization by creating or contributing to a software security policy that satisfies internal, regulatory, and customer-driven security requirements. This policy is what is permitted and denied at the initiative level—if it's not mandatory and enforced, it's not policy. It includes a unified approach for satisfying the (potentially lengthy) list of security drivers at the governance level so project teams can avoid keeping up with the details involved in complying with all applicable regulations or other mandates. Likewise, project teams won't need to relearn customer security requirements on their own. Architecture standards and coding guidelines aren't examples of policy, but policy that prescribes and mandates their use for certain software categories falls under that umbrella. In many cases, policy statements are translated into automation to provide governance-as-code. Even if not enforced by humans, policy that's been automated must still be mandatory. In some cases, policy will be documented exclusively as governance-as-code (see [SM3.4]), often as tool configuration, but it must still be readily readable, auditable, and editable by humans.

[CP2.1: 58] Build a PII inventory.

The organization identifies and tracks the kinds of PII processed or stored by each of its systems, along with their associated data

repositories. In general, simply noting which applications process PII isn't enough—the type of PII (e.g., PHI, PFI, PI, etc.) and where it is stored are necessary so the inventory can be easily referenced in critical situations. This usually includes making a list of databases that would require customer notification if breached or a list to use in crisis simulations (see [CMVM3.3]). Build the PII inventory by starting with each individual application and noting its PII use, or by starting with PII types and noting the applications that touch each type. System architectures have evolved such that PII will flow into cloud-based service and endpoint device ecosystems, then come to rest there (e.g., content delivery networks, social networks, mobile devices, IoT devices), making it tricky to keep an accurate PII inventory.

[CP2.2: 59] Require security sign-off for compliance-related risk.

The organization has a formal compliance risk acceptance sign-off and accountability process that addresses all software development projects. In this process, the SSG acts as an advisor while the risk owner signs off on the software's compliance state prior to release based on its adherence to documented criteria. The sign-off policy might also require the head of the business unit to acknowledge compliance issues that haven't been mitigated or compliance-related SSDL steps that have been skipped, but it is required even when no compliance-related risk is present. Sign-off is explicit and captured for future reference, with any exceptions tracked, even in automated application lifecycle methodologies. Note that an application without security defects might still be noncompliant, so clean security testing results are not a substitute for a compliance sign-off. Even in DevOps organizations where engineers have the technical ability to release software, there is still a need for a deliberate risk acceptance step even if the compliance criteria are embedded in automation (see [SM3.4]). In cases where the risk owner signs off on a particular set of compliance acceptance criteria that are then implemented in automation to provide governance-as-code, there must be ongoing verification that the criteria remain accurate, and the automation is actually working.

[CP2.3: 73] Implement and track controls for compliance.

The organization can demonstrate compliance with applicable requirements because its SSDL is aligned with the control statements developed by the SSG in collaboration with compliance stakeholders (see [CP1.1]). The SSG collaborates with stakeholders to track controls, navigate problem areas, and ensure that auditors and regulators are satisfied. The SSG can remain in the background when the act of following the SSDL automatically generates the desired compliance evidence predictably and reliably. Increasingly, the DevOps approach embeds compliance controls in automation, such as in software-defined infrastructure and networks, rather than in human process and manual intervention. A firm doing this properly can explicitly associate satisfying its compliance concerns with following its SSDL.

[CP2.4: 62] Include software security SLAs in all vendor contracts.

Software vendor contracts include an SLA to ensure that the vendor's security efforts align with the organization's compliance story. Each new or renewed contract contains provisions requiring the vendor to address software security and deliver a product or service compatible with the organization's security policy. In some cases, open source

licensing concerns initiate the vendor management process, which can open the door for additional software security language in the SLA (see [SR2.5]). Typical provisions set requirements for policy conformance, incident management, training, defect management, and response times for addressing software security issues. Traditional IT security requirements and a simple agreement to allow penetration testing or another defect discovery activity aren't sufficient here.

[CP2.5: 82] Ensure executive awareness of compliance and privacy obligations.

Gain buy-in around compliance and privacy activities by providing executives with plain-language explanations of the organization's compliance and privacy obligations, along with the potential consequences of failing to meet those obligations. For some organizations, explaining the direct cost and likely fallout from a compliance failure or data breach can be an effective way to broach the subject. For others, having an outside expert address the Board works because some executives value an outside perspective more than an internal one. A sure sign of proper executive buy-in is an acknowledgment of the need along with adequate allocation of resources to meet those obligations. Use the sense of urgency that typically follows a compliance or privacy failure to build additional awareness and bootstrap new efforts.

[CP3.1: 30] Document a software compliance story.

The SSG can demonstrate the organization's software security compliance story on demand using a combination of written policy, controls documentation, and artifacts gathered through the SSDL. Often, senior management, auditors, and regulators—whether government or other—will be satisfied with the same kinds of reports that can be generated directly from various tools. In some cases, particularly where organizations leverage shared responsibility through cloud services, the organization will require additional information from vendors about how that vendor's controls support organizational compliance needs. It will often be necessary to normalize information that comes from disparate sources.

[CP3.2: 28] Ensure compatible vendor policies.

Ensure that vendor software security policies and SSDL processes are compatible with internal policies. Vendors likely comprise a diverse group—cloud providers, middleware providers, virtualization providers, container and orchestration providers, bespoke software creators, contractors, and many more—and each might be held to different policy requirements. Policy adherence enforcement might be through a point-in-time review (such as ensuring acceptance criteria), automated checks (such as those applied to pull requests, committed artifacts like containers, or similar), or convention and protocol (such as preventing services connection unless security settings are correct and expected certificates are present). Evidence of vendor adherence could include results from SSDL activities, from manual tests or tests built directly into automation or infrastructure, or from other software lifecycle instrumentation. For some policies or SSDL processes, vendor questionnaire responses and attestation alone might be sufficient.

[CP3.3: 11] Drive feedback from software lifecycle data back to policy.

Feed information from the software lifecycle into the policy creation and maintenance process to drive improvements, such as defect prevention and strengthening governance-as-code practices (see

[SM3.4]). With this feedback as a routine process, blind spots can be eliminated by mapping them to trends in SSDL failures. Events such as the regular appearance of inadequate architecture analysis, recurring vulnerabilities, ignored security release conditions, or the wrong vendor choice for carrying out a penetration test can expose policy weakness (see [CP1.3]). As an example, lifecycle data might indicate that policies impose too much bureaucracy by introducing friction that prevents engineering from meeting the expected delivery cadence. Rapid technology evolution might also create policy gaps that must be addressed. Over time, policies become more practical and easier to carry out (see [SM1.1]). Ultimately, policies are refined with SSDL data to enhance and improve a firm's effectiveness.

Governance: Training (T)

Training has always played a critical role in software security because organizational stakeholders across GRC, legal, engineering, operations, and other groups often start with little security knowledge.

[T1.1: 71] Conduct software security awareness training.

To promote a culture of software security throughout the organization, the SSG conducts periodic software security awareness training. This training might be delivered via SSG members, an outside firm, the internal training organization, or e-learning, but course content isn't necessarily tailored for a specific audience—developers, QA engineers, and project managers could attend the same “Introduction to Software Security” course, for example. Augment this content with a tailored approach that addresses the firm's culture explicitly, which might include the process for building security in, avoiding common mistakes, and technology topics such as CI/CD and DevSecOps. Generic introductory courses that cover basic IT or high-level security concepts don't generate satisfactory results. Likewise, awareness training aimed only at developers and not at other roles in the organization is insufficient.

[T1.7: 58] Deliver on-demand individual training.

The organization lowers the burden on students and reduces the cost of delivering software security training by offering on-demand training for SSDL stakeholders. The most obvious choice, e-learning, can be kept up to date through a subscription model, but an online curriculum must be engaging and relevant to students in various roles (e.g., developer, QA, cloud, ops, etc.) to achieve its intended purpose. Ineffective (e.g., aged, off-topic) training or training that isn't used won't create any change. Hot topics like containerization and security orchestration, and new delivery styles such as gamification, will attract more interest than boring policy discussions. For developers, it's possible to provide training directly through the IDE right when it's needed, but in some cases, building a new skill (such as cloud security or threat modeling) might be better suited for instructor-led training, which can also be provided on demand.

[T1.8: 53] Include security resources in onboarding.

The process for bringing new hires into a software engineering organization requires timely completion of a training module about software security. While the generic new hire process usually covers topics like picking a good password and avoiding phishing, this orientation period is enhanced to cover topics such as how to create, deploy, and operate secure code, the SSDL, security standards (see [SR1.1]), and internal security resources (see [SR1.2]). The objective is

to ensure that new hires contribute to the security culture as soon as possible. Although a generic onboarding module is useful, it doesn't take the place of a timely and more complete introductory software security course.

[T2.5: 38] Enhance satellite (security champions) through training and events.

Strengthen the satellite network (see [SM2.3]) by inviting guest speakers or holding special events about advanced software security topics. This effort is about providing to the satellite customized training (e.g., the latest software security techniques for DevOps or serverless technologies, or on the implications of new policies and standards) so that it can fulfill its assigned responsibilities—it's not about inviting satellite members to routine brown bags or signing them up for standard computer-based training. Similarly, a standing conference call with voluntary attendance won't get the desired results, which are as much about building camaraderie as they are about sharing knowledge and organizational efficiency. Regular events build community and facilitate collaboration and collective problem-solving. Face-to-face meetings are by far the most effective, even if they happen only once or twice a year and even if some participants must attend by videoconferencing. In teams with many geographically dispersed and work-from-home members, simply turning on cameras and ensuring that everyone gets a chance to speak makes a substantial difference.

[T2.8: 28] Create and use material specific to company history.

To make a strong and lasting change in behavior, training includes material specific to the company's history of software security challenges. When participants can see themselves in a problem, they're more likely to understand how the material is relevant to their work as well as when and how to apply what they've learned. One way to do this is to use noteworthy attacks on the company's software as examples in the training curriculum. Both successful and unsuccessful attacks, as well as notable results from penetration tests and red team exercises, can make good teachable moments. Stories from company history can help steer training in the right direction, but only if those stories are still relevant and not overly censored. This training should cover platforms used by developers (developers orchestrating containers probably won't care about old virtualization problems) and problems relevant to languages in common use.

[T2.9: 33] Deliver role-specific advanced curriculum.

Software security training goes beyond building awareness (see [T1.1]) by enabling students to incorporate security practices into their work. This training is tailored to cover the tools, technology stacks, development methodologies, and issues that are most relevant to students. An organization could offer tracks for its engineers, for example, one each for architects, developers, operations, DevOps, site reliability engineers, and testers. Tool-specific training is also commonly needed in such a curriculum. While it might be more concise than engineering training, role-specific training is also necessary for many stakeholders within an organization, including product management, executives, and others. In any case, the training must be taken by a broad enough audience to build the collective skillsets required.

[T2.10: 28] Host software security events.

The organization hosts security events featuring external speakers and content in order to strengthen its security culture. Good examples of such events are Intel iSecCon and AWS re:Inforce, which invite all employees, feature external presenters, and focus on helping engineering create, deploy, and operate better code. Employees benefit from hearing outside perspectives, especially those related to fast-moving technology areas with software security ramifications, and the organization benefits from putting its security credentials on display (see [SM3.2]). Events open only to small, select groups, or simply putting recordings on an internal portal, won't result in the desired culture change across the organization.

[T2.11: 27] Require an annual refresher.

Everyone involved in the SSDL is required to take an annual software security refresher course. This course keeps the staff up to date on the organization's security approach and ensures that the organization doesn't lose focus due to turnover, evolving methodologies, or changing deployment models. The SSG might give an update on the security landscape and explain changes to policies and standards. A refresher could also be rolled out as part of a firm-wide security day or in concert with an internal security conference. Coverage of new topics and changes to the previous year's content should result in a significant amount of fresh content.

[T3.1: 9] Reward progression through curriculum.

Progression through the security curriculum brings personal benefits, such as public acknowledgement or career advancement. The reward system can be formal and lead to a certification or an official mark in the human resources system, or it can be less formal and include motivators such as documented praise at annual review time. Involving a corporate training department and human resources team can make the impact of improving security skills on career progression more obvious, but the SSG should continue to monitor security knowledge in the firm and not cede complete control or oversight. Coffee mugs and t-shirts can build morale, but it usually takes the possibility of real career progression to change behavior.

[T3.2: 16] Provide training for vendors and outsourced workers.

Vendors and outsourced workers receive the same level of software security training given to employees. Spending time and effort helping suppliers get security right at the outset is much easier than trying to determine what went wrong later, especially if the development team has moved on to other projects. Training individual contractors is much more natural than training entire outsourced firms and is a reasonable place to start. It's important that everyone who works on the firm's software has an appropriate level of training that increases their capability of meeting the software security expectations for their role, regardless of their employment status. Of course, some vendors and outsourced workers might have received adequate training from their own firms, but that should always be verified.

[T3.5: 22] Provide expertise via open collaboration channels.

Software security experts offer help to anyone in an open manner during regularly scheduled office hours or openly accessible channels on Slack, Jira, or similar. By acting as an informal resource for people who want to solve security problems, the SSG leverages teachable

moments and emphasizes the carrot over the stick approach to security best practices. Office hours might be hosted one afternoon per week by a senior SSG member, perhaps inviting briefings from product or application groups working on hard security problems. Slack and other messaging applications can capture questions 24x7, functioning as an office hours platform when appropriate subject matter experts are consistently part of the conversation and are ensuring that the answers generated align with SSG expectations. An online approach has the added benefit of discussions being recorded and searchable.

[T3.6: 7] Identify new satellite members (security champions) through observation.

Future satellite members (e.g., security champions) are recruited by noting people who stand out during training courses, office hours, capture-the-flag exercises, hack-a-thons, and other opportunities that show skill and enthusiasm, then encouraging them to join the satellite. Pay particular attention to practitioners who are contributing things such as code, security configurations, or defect discovery rules. The satellite often begins as an assigned collection of people scattered across the organization who show an above-average level of security interest or advanced knowledge of new technology stacks and development methodologies (see [SM2.3]). Identifying future members proactively is a step toward creating a social network that speeds the adoption of security into software development and operations. A group of enthusiastic and skilled volunteers will be easier to lead than a group that is drafted.

INTELLIGENCE

Intelligence: Attack Models (AM)

Attack Models capture information used to think like an attacker, including threat modeling inputs, abuse cases, data classification, and technology-specific attack patterns.

[AM1.2: 80] Use a data classification scheme for software inventory.

Security stakeholders in an organization agree on a data classification scheme and use it to inventory software, delivery artifacts (e.g., containers), and associated persistent data stores according to the kinds of data processed or services called, regardless of deployment model (e.g., on- or off-premises). Many classification schemes are possible—one approach is to focus on PII, for example. Depending on the scheme and the software involved, it could be easiest to first classify data repositories (see [CP2.1]), then derive classifications for applications according to the repositories they use. Other approaches include data classification according to protection of intellectual property, impact of disclosure, exposure to attack, relevance to GDPR, and geographic boundaries.

[AM1.3: 42] Identify potential attackers.

The SSG identifies potential attackers in order to understand their motivations and abilities. The outcome of this periodic exercise could be a set of attacker profiles that includes outlines for categories of attackers and more detailed descriptions for noteworthy individuals that are used in end-to-end design review (see [AA1.2]). In some cases, a third-party vendor might be contracted to provide this information. Specific and contextual attacker information is almost always more useful than generic

information copied from someone else's list. Moreover, a list that simply divides the world into insiders and outsiders won't drive useful results. Identification of attackers should also consider the organization's evolving software supply chain, attack surface, theoretical internal attackers, and contract staff.

[AM1.5: 76] Gather and use attack intelligence.

The SSG ensures the organization stays ahead of the curve by learning about new types of attacks and vulnerabilities, then adapts that information to the organization's needs. Attack intelligence must be made actionable and useful for a variety of consumers, which might include developers, testers, DevOps, security operations, and reliability engineers, among others. In many cases, a subscription to a commercial service can provide a reasonable way of gathering basic attack intelligence related to applications, APIs, containerization, orchestration, cloud environments, and so on. Attending technical conferences and monitoring attacker forums, then correlating that information with what's happening in the organization (perhaps by leveraging automation to mine operational logs and telemetry) helps the SSG learn more about emerging vulnerability exploitation.

[AM2.1: 16] Build attack patterns and abuse cases tied to potential attackers.

The SSG works with stakeholders to build attack patterns and abuse cases tied to potential attackers (see [AM1.3]). These resources can be built from scratch or from standard sets, such as the MITRE ATT&CK framework, and the SSG adds to the pile based on its own attack stories to prepare the organization for SSDL activities such as design review and penetration testing. For example, a story about an attack against a poorly designed cloud-native application could lead to a containerization attack pattern that drives a new type of testing (see [ST3.5]). If a firm tracks the fraud and monetary costs associated with specific attacks, this information can in turn be used to prioritize the process of building attack patterns and abuse cases. Organizations will likely need to evolve their attack pattern and abuse case creation prioritization and content over time due to changing software architectures (e.g., zero trust, cloud-native, serverless), attackers, and technologies.

[AM2.2: 11] Create technology-specific attack patterns.

The SSG facilitates technology-specific attack pattern creation by collecting and providing knowledge about attacks relevant to the organization's technologies. For example, if the organization's cloud software relies on a cloud vendor's security apparatus (e.g., key and secrets management), the SSG can help catalog the quirks of the crypto package and how it might be exploited. Attack patterns directly related to the security frontier (e.g., serverless) can be useful here as well. It's often easiest to start with existing generalized attack patterns to create the needed technology-specific ones, but simply adding "for microservices" at the end of a generalized pattern name, for example, won't suffice.

[AM2.5: 16] Maintain and use a top N possible attacks list.

The SSG periodically digests the ever-growing list of attack types, creates a prioritized short list—the top N—and then uses the list to drive change. This initial list almost always combines input from multiple sources, both inside and outside the organization. Some organizations prioritize their list according to a perception of potential business loss while others might prioritize according to preventing

successful attacks against their software. The top N list doesn't need to be updated with great frequency, and attacks can be coarsely sorted. For example, the SSG might brainstorm twice a year to create lists of attacks the organization should be prepared to counter "now," "soon," and "someday."

[AM2.6: 16] Collect and publish attack stories.

To maximize the benefit from lessons that don't always come cheap, the SSG collects and publishes stories about attacks against the organization's software. Both successful and unsuccessful attacks can be noteworthy, and discussing historical information about software attacks has the added effect of grounding software security in a firm's reality. This is particularly useful in training classes (see [T2.8]) to help counter a generic approach that might be overly focused on other organizations' most common bug lists or outdated platform attacks. Hiding or overly sanitizing information about attacks from people building new systems fails to garner any positive benefits from a negative event.

[AM2.7: 14] Build an internal forum to discuss attacks.

The organization has an internal, interactive forum where the SSG, the satellite, incident response, and others discuss attacks and attack methods. The discussion serves to communicate the attacker perspective to everyone. It's useful to include all successful attacks here, regardless of attack source, such as supply chain, internal, consultants, or bug bounty contributors. The SSG augments the forum with an internal communications channel (see [T3.5]) that encourages subscribers to discuss the latest information on publicly known incidents. Dissection of attacks and exploits that are relevant to a firm are particularly helpful when they spur discussion of development, infrastructure, and other mitigations. Simply republishing items from public mailing lists doesn't achieve the same benefits as active and ongoing discussions, nor does a closed discussion hidden from those creating code and configurations. Everyone should feel free to ask questions and learn about vulnerabilities and exploits.

[AM3.1: 9] Have a research group that develops new attack methods.

A research group works to identify and mitigate the impact of new classes of attacks and shares their knowledge with stakeholders. Identification does not always require original research—the group might expand on an idea discovered by others. Doing this research in-house is especially important for early adopters of new technologies and configurations so that they can discover potential weaknesses before attackers do. One approach is to create new attack methods that simulate persistent attackers during goal-oriented red team exercises (see [PT3.1]). This isn't a penetration testing team finding new instances of known types of weaknesses, it's a research group that innovates attack methods and mitigation approaches. Example mitigation approaches include test cases, static analysis rules, attack patterns, standards, and policy changes. Some firms provide researchers time to follow through on their discoveries by using bug bounty programs or other means of coordinated disclosure (see [CMVM3.7]). Others allow researchers to publish their findings at conferences like DEF CON to benefit everyone.

[AM3.2: 5] Create and use automation to mimic attackers.

The SSG arms engineers, testers, and incident response with automation to mimic what attackers are going to do. For example, a new attack method identified by an internal research group (see [AM3.1]) or a disclosing third party could require a new tool, so the SSG could package the tool and distribute it to testers. The idea here is to push attack capability past what typical commercial tools and offerings encompass, then make that knowledge and technology easy for others to use. Mimicking attackers, and especially attack chains, almost always requires tailoring tools to a firm's particular technology stacks, infrastructure, and configurations. When technology stacks and coding languages evolve faster than vendors can innovate, creating tools and automation in-house might be the best way forward. In the DevOps world, these tools might be created by engineering and embedded directly into toolchains and automation (see [ST3.6]).

[AM3.3: 11] Monitor automated asset creation.

Implement technology controls that provide a continuously updated view of the various network, machine, software, and related infrastructure assets being instantiated by engineering teams. To help ensure proper coverage, the SSG works with engineering teams (including potential shadow IT teams) to understand orchestration, cloud configuration, and other self-service means of software delivery to ensure proper monitoring. This monitoring requires a specialized effort—normal system, network, and application logging and analysis won't suffice. Success might require a multi-pronged approach, including consuming orchestration and virtualization metadata, querying cloud service provider APIs, and outside-in web crawling and scraping.

Intelligence: Security Features & Design (SFD)

The Security Features & Design practice is charged with creating usable security patterns for major security controls (meeting the standards defined in the Standards & Requirements practice), building components and services for those controls, and establishing collaboration during security design efforts.



[SFD1.1: 104] Integrate and deliver security features.

Provide proactive guidance on preapproved security features for engineering groups to use rather than each group implementing its own security features. Engineering groups benefit from implementations that come preapproved, and the SSG benefits by not having to repeatedly track down the kinds of subtle errors that often creep into security features (e.g., authentication, role management, key management, logging, cryptography, protocols). These features might be discovered during SSDL activities, created by the SSG or specialized development teams, or defined in configuration templates (e.g., cloud blueprints) and delivered via mechanisms such as containers, microservices, and APIs. Generic security features often must be tailored for specific platforms. For example, each mobile and cloud platform will likely need its own means by which users are authenticated and authorized, secrets are managed, and user actions are centrally logged and monitored. It's implementing these defined security features that generates real progress, not simply making a list of them.

[SFD1.2: 90] Application architecture teams engage with the SSG.

Application architecture teams take responsibility for security in the same way they take responsibility for performance, availability, scalability, and resiliency. One way to keep security from falling out of these architecture discussions is to have secure design experts (from the SSG, a vendor, etc.) participate. Increasingly, architecture discussions include developers and site reliability engineers who are governing all types of software components, such as open source, APIs, containers, and cloud services. In other cases, enterprise architecture teams have the knowledge to help the experts create secure designs that integrate properly into corporate design standards. Proactive engagement with experts is key to success here. In addition, it's never safe for one team to assume another team has addressed security requirements—even moving a well-known system to the cloud means reengaging the experts.

[SFD2.1: 39] Leverage secure-by-design components and services.

Build or provide approved secure-by-design software components and services for use by engineering teams. Prior to approving and publishing secure-by-design software components and services, including open source and cloud services, the SSG must carefully assess them for security. This assessment process to declare a component secure-by-design is usually more rigorous and in-depth than that for typical projects. In addition to teaching by example, these resilient and reusable building blocks aid important efforts such as architecture analysis and code review by making it easier to avoid mistakes. These components and services also often have features (e.g., application identity, RBAC) that enable uniform usage across disparate environments. Similarly, the SSG might further take advantage of this defined list by tailoring static analysis rules specifically for the components it offers (see [CR2.6]).

[SFD2.2: 64] Create capability to solve difficult design problems.

Contribute to building resilient architectures by solving design problems unaddressed by organizational security components or services, or by cloud service providers, thus minimizing the negative impact that security has on other constraints, such as feature velocity. Involving the SSG and secure design experts in application refactoring or in the design of a new protocol, microservice, or architecture decision (e.g., containerization) enables timely analysis of the security implications of existing defenses and identifies elements to be improved. Designing for security early in the new project process is more efficient than analyzing an existing design for security and then refactoring when flaws are uncovered (see [AA1.1], [AA1.2], [AA2.1]). The SSG could also get involved in what would have historically been purely engineering discussions, as even rudimentary use of cloud-native technologies (e.g., "Hello, world!") requires proper use of configurations and other capabilities that have direct implications on security posture.

[SFD3.1: 17] Form a review board to approve and maintain secure design patterns.

A review board formalizes the process of reaching and maintaining consensus on security tradeoffs in design needs. Unlike a typical architecture committee focused on functions, this group focuses on providing security guidance, often in the form of

patterns, standards, features, or frameworks. It also periodically reviews already published design guidance (especially around authentication, authorization, and cryptography) to ensure that design decisions don't become stale or out of date. This review board helps control the chaos associated with adoption of new technologies when development groups might otherwise make decisions on their own without engaging the SSG. Review board security guidance can also serve to inform outsourced software providers about security expectations (see [CP3.2]).

[SFD3.2: 18] Require use of approved security features and frameworks.

Implementers must take their security features and frameworks from an approved list or repository (see [SFD1.1], [SFD2.1], [SFD3.1]). There are two benefits to this activity—developers don't spend time reinventing existing capabilities, and review teams don't have to contend with finding the same old defects in new projects or when new platforms are adopted. Reusing proven components eases testing, code review, and threat modeling (see [AA1.1]). Reuse is a major advantage of consistent software architecture and is particularly helpful for Agile development and velocity maintenance in CI/CD pipelines. Packaging and applying required components, such as via containerization (see [SE2.5]), makes it especially easy to reuse approved features and frameworks.

[SFD3.3: 7] Find and publish secure design patterns from the organization.

Foster centralized design reuse by collecting secure design patterns (sometimes referred to as security blueprints) from across the organization and publishing them for everyone to use. A section of the SSG website (see [SR1.2]) could promote positive elements identified during threat modeling or architecture analysis so that good ideas spread widely. This process is formalized—an ad hoc, accidental noticing isn't sufficient. Common design patterns accelerate development, so it's important to use secure design patterns, not just for applications but for all software assets (e.g., microservices, APIs, containers, infrastructure, and automation).

Intelligence: Standards & Requirements (SR)

The Standards & Requirements practice involves eliciting explicit software security requirements from the organization, determining which COTS tools to recommend, building standards for major security controls (such as authentication, input validation, and so on), creating security standards for technologies in use, and creating a standards review process.

[SR1.1: 96] Create security standards.

The organization meets the demand for security guidance by creating standards that explain the required way to adhere to policy and carry out security-centric design, development, and operations. A standard might describe how to perform identity-based application authentication or how to implement transport-level security, perhaps with the SSG ensuring the availability of a reference implementation. Standards often apply to software beyond the scope of an application's code, including container construction, orchestration, infrastructure-as-code, and cloud security configuration. Standards can be deployed in a variety of ways to keep them actionable and relevant. They can be automated into development environments (such as an IDE or toolchain) or

explicitly linked to code examples and deployment artifacts (e.g., containers). In any case, to be considered standards, they must be adopted and enforced.

[SR1.2: 101] Create a security portal.

The organization has a well-known central location for information about software security. Typically, this is an internal website maintained by the SSG and satellite (security champions) that people refer to for current information on security policies, standards, and requirements, as well as for other resources (such as training). An interactive portal is better than a static portal with guideline documents that rarely change. Organizations often supplement these materials with mailing lists, chat channels (see [T3.5]), and face-to-face meetings. Development teams are increasingly putting software security knowledge directly into toolchains and automation that are outside the organization (e.g., GitHub), but that does not remove the need for SSG-led knowledge management.



[SR1.3: 103] Translate compliance constraints to requirements.

Compliance constraints are translated into security requirements for individual projects and communicated to the engineering teams. This is a linchpin in the organization's compliance strategy—by representing compliance constraints explicitly with requirements and informing stakeholders, the organization demonstrates that compliance is a manageable task. For example, if the organization builds software that processes credit card transactions, PCI DSS compliance plays a role during the security requirements phase. In other cases, technology standards built for international interoperability can include security guidance on compliance needs. Representing these standards as requirements also helps with traceability and visibility in the event of an audit. It's particularly useful to codify the requirements into reusable code (see [SFD2.1]) or artifact deployment specifications (see [SE2.2]).

[SR2.2: 80] Create a standards review process.

Create a process to develop software security standards and ensure that all stakeholders have a chance to weigh in. This review process could operate by appointing a spokesperson for any proposed security standard, putting the onus on the person to demonstrate that the standard meets its goals and to get buy-in and approval from stakeholders. Enterprise architecture or enterprise risk groups sometimes take on the responsibility of creating and managing standards review processes. When the standards are implemented directly as software, the responsible person might be a DevOps manager, release engineer, or whoever owns the associated deployment artifact (e.g., the orchestration code).

[SR2.4: 92] Identify open source.

Identify open source components and dependencies included in the organization's code repositories and built software, then review them to understand their security posture. Organizations use a variety of tools and metadata provided by delivery pipelines to discover old versions of open source components with known vulnerabilities or that their software relies on multiple versions of the same component. Scale efforts by using automated tools to find open source, whether whole components or perhaps large chunks of borrowed code. Some software development pipeline platforms, container registries, and middleware platforms have begun to provide

this visibility as metadata (e.g., SBOMs, [SE3.6]) resulting from behind-the-scenes artifact scanning. Some organizations combine composition analysis results from multiple phases of the software lifecycle to get a more complete and accurate list of the open source being included in production software.

[SR2.5: 63] Create SLA boilerplate.

The SSG works with the legal department to create standard SLA boilerplate for use in contracts with vendors and outsource providers, including cloud providers, to require software security efforts on their part. The legal department might also leverage the boilerplate to help prevent compliance and privacy problems. Under the agreement, vendors and outsource providers must meet company-mandated software security SLAs (see [CP2.4]). Boilerplate language might call for objective third-party insight into software security efforts, such as BSIMMsc measurements or BSIMM scores.

[SR2.7: 53] Control open source risk.

The organization has control over its exposure to the risks that come along with using open source components and all the involved dependencies, including dependencies integrated at runtime. Controlling exposure usually includes multiple efforts, with one example being responding to known vulnerabilities in identified open source (see [SR2.4]). The use of open source could also be restricted to predefined projects or to a short list of versions that have been through an approved security screening process, have had unacceptable vulnerabilities remediated, and are made available only through approved internal repositories and containers. For some use cases, policy might preclude any use of open source. The legal department often spearheads additional open source controls due to the viral license problem associated with GPL code. SSGs that partner with and educate the legal department can help move an organization to improve its open source risk management practices, which must be applied across the software portfolio to be effective.

[SR3.2: 19] Communicate standards to vendors.

Work with vendors to educate them and promote the organization's security standards. A healthy relationship with a vendor isn't guaranteed through contract language alone (see [CP2.4]), so the SSG should engage with vendors, discuss vendor security practices, and explain in simple terms (rather than legalese) what the organization expects. Any time a vendor adopts the organization's security standards, it's a clear sign of progress. Note that standards implemented as security features or infrastructure configuration could be a requirement to services integration with a vendor (see [SFD1.1], [SE2.2]). When the firm's SSDL is publicly available, communication regarding software security expectations is easier. Likewise, sharing internal practices and measures can make expectations clear.

[SR3.3: 17] Use secure coding standards.

Developers use secure coding standards to avoid the most obvious bugs and as ground rules for code review. These standards are necessarily specific to a programming language, and they can address the use of popular frameworks, APIs, libraries, and infrastructure automation. Secure coding standards can also be for low- or no-code platforms (e.g., Microsoft Power Apps, Salesforce Lightning). While enforcement isn't the point at this stage (see [CR3.5]), violation of standards is a teachable moment

for all stakeholders. Other useful coding standards topics include proper use of cloud APIs, use of approved cryptography, memory sanitization, banned functions, open source use, and many others. If the organization already has coding standards for other purposes, its secure coding standards should build upon them. A clear set of secure coding standards is a good way to guide both manual and automated code review, as well as to provide relevant examples for security training. Some groups might choose to integrate their secure coding standards directly into automation. Socializing the benefits of following standards is also a good first step to gaining widespread acceptance (see [SM2.7]).

[SR3.4: 19] Create standards for technology stacks.

The organization standardizes on the use of specific technology stacks. This translates into a reduced workload because teams don't have to explore new technology risks for every new project. The organization might create a secure base configuration (commonly in the form of golden images, Terraform definitions, etc.) for each technology stack, further reducing the amount of work required to use the stack safely. In cloud environments, hardened configurations likely include up-to-date security patches, security configuration, and security services, such as logging and monitoring. In traditional on-premises IT deployments, a stack might include an operating system, a database, an application server, and a runtime environment (e.g., a LAMP stack). Standards for secure use of reusable technologies, such as containers, microservices, or orchestration code, means that getting security right in one place positively impacts the security posture of all downstream efforts (see [SE2.5]).

SDLC TOUCHPOINTS

SDLC Touchpoints: Architecture Analysis (AA)

Architecture analysis encompasses capturing software architecture in concise diagrams, applying lists of risks and threats, adopting a process for review (such as Microsoft Threat Modeling [STRIDE] or Architecture Risk Analysis [ARA]), building an assessment and remediation plan for the organization, and using a risk methodology to rank applications.



[AA1.1: 113] Perform security feature review.

Security-aware reviewers identify application security features, review these features against application security requirements and runtime parameters, and determine if each feature can adequately perform its intended function—usually referred to as threat modeling. The goal is to quickly identify missing security features and requirements, or bad deployment configuration (authentication, access control, use of cryptography, etc.), and address them. For example, threat modeling would identify both a system that was subject to escalation of privilege attacks because of broken access control as well as a mobile application that incorrectly puts PII in local storage. Use of the firm's secure-by-design components often streamlines this process (see [SFD2.1]). Many modern applications are no longer simply "3-tier" but instead involve components architected to interact across a variety of tiers—browser/endpoint, embedded, web, microservices, orchestration engines, deployment pipelines, third-party SaaS, and so on. Some of these environments might provide robust security feature sets, whereas others might have key capability gaps that

require careful analysis, so organizations should consider the applicability and correct use of security features across all tiers that constitute the architecture and operational environment.

[AA1.2: 53] Perform design review for high-risk applications.

Perform a design review to determine whether security features and deployment configuration are resistant to attack in an attempt to break the design. The goal is to extend the more formulaic approach of a security feature review (see [AA1.1]) to model application behavior in the context of real-world attackers and attacks. Reviewers must have some experience beyond simple threat modeling to include performing detailed design reviews and breaking the design under consideration. Rather than security feature guidance, a design review should produce a set of flaws and a plan to mitigate them. An organization can use consultants to do this work, but it should participate actively. A review focused only on whether a software project has performed the right process steps won't generate useful results about flaws. Note that a sufficiently robust design review process can't be executed at CI/CD speed, so organizations should focus on a few high-risk applications to start (see [AA1.4]).

[AA1.4: 69] Use a risk methodology to rank applications.

Use a defined risk methodology to collect information about each application in order to assign a risk classification and associated prioritization. It is important to use this information in prioritizing what applications or projects are in scope for testing, including security feature and design reviews. Information collection can be implemented via questionnaire or similar method, whether manual or automated. Information needed for classification might include, "Which programming languages is the application written in?" or "Who uses the application?" or "Is the application's deployment software-orchestrated?" Typically, a qualified member of the application team provides the information, but the process should be short enough to take only a few minutes. The SSG can use the answers to categorize the application as, for example, high, medium, or low risk. Because a risk questionnaire can be easy to game, it's important to put into place some spot-checking for validity and accuracy—an overreliance on self-reporting can render this activity useless.

[AA2.1: 31] Perform architecture analysis using a defined process.

Define and use a process for architecture analysis (AA) that extends the design review (see [AA1.2]) to also document business risk in addition to technical flaws. The goal is to identify application design flaws as well as the associated risk (e.g., impact of exploitation), such as through frequency or probability analysis, to properly inform stakeholder risk management efforts. The AA process includes a standardized approach for thinking about attacks, vulnerabilities, and various security properties. The process is defined well enough that people outside the SSG can carry it out. It's important to document both the architecture under review and any security flaws uncovered, as well as risk information that people can understand and use. Microsoft Threat Modeling, Versprite PASTA, and Synopsys ARA are examples of such a process, although these will likely need to be tailored to a given environment. In some cases, performing AA and documenting business risk is done by different teams working together in a single process. Uncalibrated or ad hoc AA approaches don't count as a defined process.

[AA2.2: 32] Standardize architectural descriptions.

Threat modeling, design review, or AA processes use an agreed-upon format (e.g., diagramming language and icons, not a Word document template) to describe architecture, including a means for representing data flow. Standardizing architecture descriptions between those who generate the models and those who analyze and annotate them makes analysis more tractable and scalable. High-level network diagrams, data flow, and authorization flows are always useful, but the model should also go into detail about how the software itself is structured. A standard architecture description can be enhanced to provide an explicit picture of information assets that require protection, including useful metadata. Standardized icons that are consistently used in diagrams, templates, and dry-erase board squiggles are especially useful, too.

[AA2.4: 38] Have SSG lead design review efforts.

The SSG takes a lead role in performing design review (see [AA1.2]) to uncover flaws. Breaking down an architecture is enough of an art that the SSG, or other reviewers outside the application team, must be proficient, and proficiency requires practice. This practice might then enable, for example, champions to take the day-to-day lead while the SSG maintains leadership around knowledge and process. The SSG can't be successful on its own, either—it will likely need help from architects or implementers to understand the design. With a clear design in hand, the SSG might be able to carry out a detailed review with a minimum of interaction with the project team. Approaches to design review evolve over time, so it's wise to not expect to set a process and use it forever. Outsourcing design review might be necessary, but it's also an opportunity to participate and learn.

[AA3.1: 20] Have engineering teams lead AA process.

Engineering teams lead AA to uncover technical flaws and document business risk. This effort requires a well-understood and well-documented process (see [AA2.1]). Even with a good process, consistency is difficult to attain because breaking architecture requires experience, so be sure to provide architects with SSG or outside expertise in an advisory capacity. Engineering teams performing AA might normally have responsibilities such as development, DevOps, cloud security, operations security, security architecture, or a variety of similar roles. The process is more useful if the AA team is different from the design team.

[AA3.2: 4] Drive analysis results into standard design patterns.

Failures identified during threat modeling, design review, or AA are fed back to security and engineering teams so that similar mistakes can be prevented in the future through improved design patterns, whether local to a team or formally approved for everyone (see [SFD3.1]). This typically requires a root-cause analysis process that determines the cause of security flaws, searches for the process that should have prevented the flaw, and makes the necessary improvements in documented security design patterns. Note that security design patterns can interact in surprising ways that break security, so apply analysis processes even when vetted design patterns are in standard use. For cloud services, providers have learned a lot about how their platforms and services fail to resist attack and have codified this experience into patterns for secure use. Organizations that heavily rely on these services might base their application-layer patterns on those building blocks provided by the cloud service provider (for example, AWS CloudFormation and Azure Blueprints) when making their own.

[AA3.3: 15] Make the SSG available as an AA resource or mentor.

To build organizational AA capability, the SSG advertises experts as resources or mentors for teams using the AA process (see [AA2.1]). This effort might enable, for example, security champions, site reliability engineers, DevSecOps engineers, and others to take the lead while the SSG offers advice. As one example, mentors help tailor AA process inputs (such as design or attack patterns) to make them more actionable for specific technology stacks. This reusable guidance helps protect the team's time so they can focus on the problems that require creative solutions rather than enumerating known bad habits. While the SSG might answer AA questions during office hours (see [T3.5]), they will often assign a mentor to work with a team, perhaps comprising both security-aware engineers and risk analysts, for the duration of the analysis. In the case of high-risk software, the SSG should play a more active mentorship role in applying the AA process.

SDLC Touchpoints: Code Review (CR)

The Code Review practice includes use of code review tools (e.g., static analysis), development of tailored rules, customized profiles for tool use by different roles (for example, developers vs. auditors), manual analysis, and tracking and measuring results.

[CR1.2: 83] Perform opportunistic code review.

Perform code review for high-risk applications in an opportunistic fashion. Organizations can follow up a design review with a code review looking for security issues in source code and dependencies, and perhaps also in deployment artifact configuration (e.g., containers) and automation metadata (e.g., infrastructure-as-code). This informal targeting often evolves into a systematic approach. Manual code review could be augmented with the use of specific tools and services, but it has to be part of a proactive process. When new technologies pop up, new approaches to code review might become necessary.



[CR1.4: 107] Use automated code review tools.

Incorporate static analysis into the code review process to make the review more efficient and consistent. Automation won't replace human judgement, but it does bring definition to the review process and security expertise to reviewers who typically aren't security experts. Note that a specific tool might not cover an entire portfolio, especially when new languages are involved, so additional local effort might be useful. Some organizations might progress to automating tool use by instrumenting static analysis into source code management workflows (e.g., pull requests) and delivery pipeline workflows (build, package, and deploy) to make the review more efficient, consistent, and in line with release cadence. Whether use of automated tools is to review a portion of the source code incrementally, such as a developer committing new code or small changes, or to conduct full analysis by scanning the entire codebase, this service should be explicitly connected to a larger SSDL defect management process applied during software development. This effort is not useful when done just to "check the security box" on the path to deployment.

[CR1.5: 62] Make code review mandatory for all projects.

A security-focused code review is mandatory for all software projects, with a lack of code review or unacceptable results stopping a release, slowing it down, or causing it to be recalled. While all

projects must undergo code review, the process might be different for different kinds of projects. The review for low-risk projects might rely more heavily on automation (see [CR1.4]), for example, whereas high-risk projects might have no upper bound on the amount of time spent by reviewers. Having a minimum acceptable standard forces projects that don't pass to be fixed and reevaluated. A code review tool with nearly all the rules turned off (so it can run at CI/CD automation speeds, for example) won't provide sufficient defect coverage. Similarly, peer code review or tools focused on quality and style won't provide useful security results.

[CR1.7: 54] Assign code review tool mentors.

Mentors show developers how to get the most out of code review tools, including configuration, triage, and remediation. Security champions, DevOps and site reliability engineers, and SSG members often make good mentors. Mentors could use office hours or other outreach to help developers establish the right configuration and get started on interpreting and remediating results. Alternatively, mentors might work with a development team for the duration of the first review they perform. Centralized use of a tool can be distributed into the development organization or toolchains over time through the use of tool mentors, but providing installation instructions and URLs to centralized tool downloads isn't the same as mentoring. Increasingly, mentorship extends to code review tools associated with deployment artifacts (e.g., container security) and infrastructure (e.g., cloud configuration).

[CR2.6: 28] Use custom rules with automated code review tools.

Create and use custom rules in code review tools to help uncover security defects specific to the organization's coding standards, or to the framework-based or cloud-provided middleware it uses. The same group that provides tool mentoring (see [CR1.7]) will likely spearhead this customization. Custom rules are often explicitly tied to proper usage of technology stacks in a positive sense and avoidance of errors commonly encountered in a firm's codebase in a negative sense. Custom rules are also an easy way to check for adherence to coding standards (see [CR3.5]). To reduce the workload for everyone, many organizations also create rules to remove repeated false positives and to turn off checks that aren't relevant.

[CR2.7: 20] Use a top N bugs list (real data preferred).

Maintain a living list of the most important kinds of bugs the organization wants to eliminate from its code and use it to drive change. Many organizations start with a generic list pulled from public sources, but broad-based lists such as the OWASP Top 10 rarely reflect an organization's bug priorities. The list's value comes from being specific to the organization, being built from real data gathered from code review (see [CR2.8]), testing (see [PT1.2]), software composition analysis (see [SE3.8]), and actual incidents (see [CMVM1.1]), then being prioritized for prevention efforts. Simply sorting the day's bug data by number of occurrences won't produce a satisfactory list because the data changes so often. To increase interest, the SSG can periodically publish a "most wanted" report after updating the list. One potential pitfall with a top N list is that it tends to include only known problems. Of course, just building the list won't accomplish anything—everyone has to use it to find and fix bugs.

[CR2.8: 34] Use centralized defect reporting to close the knowledge loop.

The bugs found during code review are tracked in a centralized repository that makes it possible to do both summary and trend reporting for the organization. The code review information is usually incorporated into a CISO-level dashboard that can include feeds from other security testing efforts (e.g., penetration testing, composition analysis, threat modeling). Given the historical code review data, the SSG can also use the reports to demonstrate progress (see [SM3.3]), then, for example, drive the training curriculum. Individual bugs make excellent training examples (see [T2.8]). Some organizations have moved toward analyzing this data and using the results to drive automation (see [ST3.6]).

[CR3.2: 14] Build a capability to combine AST results.

Combine application security testing results so that multiple testing techniques feed into one reporting and remediation process. In addition to code review, testing techniques often include dynamic analysis, software composition analysis, container scanning, cloud services configuration review, and so on. The SSG might write scripts or acquire software to gather data automatically and combine the results into a format that can be consumed by a single downstream review and reporting solution. The tricky part of this activity is normalizing vulnerability information from disparate sources that might use conflicting terminology or scoring. In some cases, using a standardized taxonomy (e.g., a CWE-like approach) can help with normalization. Combining multiple sources helps drive better-informed risk mitigation decisions.

[CR3.3: 8] Create capability to eradicate bugs.

When a security bug is found during code review (see [CR1.2], [CR1.4]), the organization searches for then fixes all occurrences of the bug, not just the instance originally discovered. Searching with custom rules (see [CR2.6]) makes it possible to eradicate the specific bug entirely without waiting for every project to reach the code review portion of its lifecycle. This doesn't mean finding every instance of every kind of cross-site scripting bug when a specific example is found—it means going after that specific example everywhere. A firm with only a handful of software applications built on a single technology stack will have an easier time with this activity than firms with many large applications built on a diverse set of technology stacks. A new development framework or library, rules in RASP or a next-generation firewall, or cloud configuration tools that provide guardrails can often help in (but not replace) eradication efforts.

[CR3.4: 2] Automate malicious code detection.

Use automated code review to identify malicious code written by in-house developers or outsource providers. Examples of malicious code include backdoors, logic bombs, time bombs, nefarious communication channels, obfuscated program logic, and dynamic code injection. Although out-of-the-box automation might identify some generic malicious-looking constructs, custom rules for the static analysis tools used to codify acceptable and unacceptable patterns in the organization's codebase will likely become a necessity. Manual review for malicious code is a good start but insufficient to complete this activity at scale. While not all backdoors or similar code were meant to be malicious when they were written (e.g.,

a developer's feature to bypass authentication during testing), such things tend to stay in deployed code and should be treated as malicious until proven otherwise. Discovering some types of malicious code will require dynamic testing techniques.

[CR3.5: 3] Enforce secure coding standards.

A violation of secure coding standards is sufficient grounds for rejecting a piece of code. This rejection can take one or more forms, such as denying a pull request, breaking a build, failing quality assurance, removal from production, or moving the code into a different development workstream where repairs or exceptions can be worked out. The enforced portions of an organization's secure coding standards (see [SR3.3]) often start out as a simple list of banned functions or required frameworks. Code review against standards must be objective—it shouldn't become a debate about whether the noncompliant code is exploitable. In some cases, coding standards are specific to language constructs and enforced with tools (e.g., codified into SAST rules). In other cases, published coding standards are specific to technology stacks and enforced during the code review process or by using automation. Standards can be positive ("do it this way") or negative ("do not use this API"), but they must be enforced.

SDLC Touchpoints: Security Testing (ST)

The Security Testing practice is concerned with prerelease defect discovery, including integrating security into standard QA processes. The practice includes the use of opaque-box application security testing (AST) tools (including fuzz testing) as a smoke test in QA, risk-driven crystal-box test suites, application of the attack model, and code coverage analysis. Security testing focuses on vulnerabilities in construction.



[ST1.1: 108] Perform edge/boundary value condition testing during QA.

QA efforts go beyond functional testing to perform basic adversarial tests and probe simple edge cases and boundary conditions, with no particular attacker skills required. When QA pushes past standard functional testing that uses expected input, it begins to move toward thinking like an adversary. Boundary value testing, whether automated or manual, can lead naturally to the notion of an attacker probing the edges on purpose (for example, determining what happens when someone enters the wrong password over and over).

[ST1.3: 97] Drive tests with security requirements and security features.

QA targets declarative security mechanisms with tests derived from security requirements and security features. A test could try to access administrative functionality as an unprivileged user, for example, or verify that a user account becomes locked after some number of failed authentication attempts. For the most part, security features can be tested in a fashion similar to other software features—security mechanisms such as account lockout, transaction limitations, entitlements, and so on are tested with both expected and unexpected input as derived from security requirements. Software security isn't security software, but testing security features is an easy way to get started. New software architectures and deployment automation, such as with container and cloud infrastructure orchestration, might require novel test approaches.

[ST1.4: 56] Integrate opaque-box security tools into the QA process.

The organization uses one or more opaque-box security testing tools as part of the QA process. Such tools are valuable because they encapsulate an attacker's perspective, albeit generically. Traditional dynamic analysis scanners are relevant for web applications, while similar tools exist for cloud environments, containers, mobile applications, embedded systems, and so on. In some situations, other groups might collaborate with the SSG to apply the tools. For example, a testing team could run the tool but come to the SSG for help with interpreting the results. When testing is integrated into Agile development approaches, opaque-box tools might be hooked into internal toolchains, provided by cloud-based toolchains, or used directly by engineering. Regardless of who runs the opaque-box tool, the testing should be properly integrated into a QA cycle of the SSDL and will often include both authenticated and unauthenticated reviews.

[ST2.4: 25] Drive QA tests with AST results.

Share results from application security testing, such as penetration testing, threat modeling, composition analysis, code reviews, etc., with QA teams to evangelize the security mindset. Using security defects as the basis for a conversation about common attack patterns or the underlying causes allows QA teams to generalize this information into new test approaches. Organizations that leverage software pipeline platforms such as GitHub, or CI/CD platforms such as the Atlassian stack, can benefit from teams receiving various testing results automatically, which should then facilitate timely stakeholder conversations—emailing security reports to QA teams will not generate the desired results. Over time, QA teams learn the security mindset, and the organization benefits from an improved ability to create security tests tailored to the organization's code.

[ST2.5: 31] Include security tests in QA automation.

Security tests are included in an automation framework and run alongside functional, performance, and other QA test suites. Executing this automation framework can be triggered manually or through additional automation (e.g., as part of pipeline tooling). When test creators who understand the software create security tests, they can uncover more specialized or more relevant defects than commercial tools might (see [ST1.4]). Security tests might be derived from typical failures of security features (see [SFD1.1]), from creative tweaks of functional and developer tests, or even from guidance provided by penetration testers on how to reproduce an issue. Tests that are performed manually or out-of-band likely will not provide timely feedback.

[ST2.6: 21] Perform fuzz testing customized to application APIs.

QA efforts include running a customized fuzzing framework against APIs critical to the organization. An API might be software that allows two applications to communicate or even software that allows a human to interact with an application (e.g., a webform). Testers could begin from scratch or use an existing fuzzing toolkit, but the necessary customization often goes beyond creating custom protocol descriptions or file format templates to giving the fuzzing framework a built-in understanding of the application interfaces and business logic. Test harnesses developed explicitly for specific applications make good places to integrate fuzz testing.

[ST3.3: 12] Drive tests with design review results.

Use design review or architecture analysis results to direct QA test creation. For example, if the results of attempting to break a design determine that “the security of the system hinges on the transactions being atomic and not being interrupted partway through” then torn transactions will become a primary target in adversarial testing. Adversarial tests like these can be developed according to a risk profile, with high-risk flaws at the top of the list. Security defect data shared with QA (see [ST2.4]) can help focus test creation on areas of potential vulnerability that can, in turn, help prove the existence of identified high-risk flaws.

[ST3.4: 4] Leverage code coverage analysis.

Testers measure the code coverage of their application security testing to identify code that isn't being exercised and then adjust test cases to incrementally improve coverage. AST can include automated testing (see [ST2.5], [ST2.6]) and manual testing (see [ST1.1], [ST1.3]). In turn, code coverage analysis drives increased security testing depth. Coverage analysis is easier when using standard measurements such as function coverage, line coverage, or multiple condition coverage. Measuring how broadly the test cases cover security requirements is not the same as measuring how broadly the test cases exercise the code. Note that standard-issue opaque-box testing tools (e.g., web application scanners, see [ST1.4]) provide exceptionally low coverage, leaving much of the software under test unexplored.

[ST3.5: 4] Begin to build and apply adversarial security tests (abuse cases).

QA teams incorporate test cases based on abuse cases (see [AM2.1]) as testers move beyond verifying functionality and take on the attacker's perspective. One way to do this is to systematically attempt to replicate incidents from the organization's history. Abuse and misuse cases based on the attacker's perspective can also be derived from security policies, attack intelligence, standards, and the organization's top N attacks list (see [AM2.5]). This effort turns the corner in QA from testing features to attempting to break the software under test.

[ST3.6: 3] Implement event-driven security testing in automation.

The SSG guides implementation of automation for continuous, event-driven application security testing. An event here is simply a noteworthy occurrence, such as dropping new code in a repository, a pull request, a build request, a push to deployment, or a Tuesday at noon. Event-driven testing implemented in pipeline automation typically moves the testing closer to the conditions driving the testing requirement (whether shift left toward design or shift right toward operations), repeats the testing as often as the event is triggered, and helps ensure that the right testing is executed for a given set of conditions. Success with this approach depends on the broad use of sensors (e.g., agents, bots) that monitor engineering processes, execute contextual rules, and provide telemetry to automation that initiates the specified testing whenever event conditions are met. More mature configurations proceed to including risk-driven conditions (e.g., size of change, provenance, function, team).

DEPLOYMENT

Deployment: Penetration Testing (PT)

The Penetration Testing practice involves standard outside-in testing of the sort carried out by security specialists. Penetration testing focuses on vulnerabilities in preproduction and production code, providing direct feeds to defect management and mitigation.



[PT1.1: 114] Use external penetration testers to find problems.

External penetration testers are used to demonstrate that the organization's software needs help. Finding critical vulnerabilities in high-profile applications provides the evidence that executives often require. Over time, the focus of penetration testing moves from trying to determine if the code is broken in some areas to a sanity check done before shipping or on a periodic basis. External penetration testers who bring a new set of experiences and skills to the problem are the most useful.

[PT1.2: 102] Feed results to the defect management and mitigation system.

All penetration testing results are fed back to engineering through established defect management or mitigation channels, with development and operations responding via a defect management and release process. In addition to application vulnerabilities, also track results from testing other software such as containers and infrastructure configuration. Properly done, this exercise demonstrates the organization's ability to improve the state of security and emphasizes the importance of not just identifying but actually fixing security problems. One way to ensure attention is to add a security flag to the bug-tracking and defect management system. The organization might leverage developer workflow or social tooling (e.g., JIRA, Slack) to communicate change requests, but these requests are still tracked explicitly as part of a vulnerability management process.

[PT1.3: 88] Use penetration testing tools internally.

The organization creates an internal penetration testing capability that uses tools as part of an established process. Execution can be part of the SSG or part of a specialized team elsewhere in the organization, with the tools complementing manual efforts to improve the efficiency and repeatability of the testing process. The tools used will usually include off-the-shelf products built specifically for application penetration testing, network penetration tools that specifically understand the application layer, container and cloud configuration testing tools, and custom scripts. Free-time or crisis-driven efforts aren't the same as an internal capability.

[PT2.2: 38] Penetration testers use all available information.

Penetration testers, whether internal or external, routinely make use of all artifacts created throughout the SSDL to do more comprehensive analysis and find more problems. Example artifacts include source code, design documents, architecture analysis results, misuse and abuse cases, code review results, and cloud environment and other deployment configurations. Focusing on high-risk applications is a good way to start. An SSDL that creates no useful artifacts about the code will make this effort harder. Having access to the artifacts is not the same as using them.

[PT2.3: 45] Schedule periodic penetration tests for application coverage.

All applications are tested periodically, which could be tied to a calendar or a release cycle. High-risk applications could get a penetration test at least once per year, for example, even if there have not been substantive code changes—other applications might receive different kinds of security testing on a similar schedule. Any security testing performed must focus on discovering vulnerabilities, not just checking a process or compliance box. This testing serves as a sanity check and helps ensure that yesterday's software isn't vulnerable to today's attacks. The testing can also help maintain the security of software configurations and environments, especially for containers and components in the cloud. One important aspect of periodic security testing across the portfolio is to make sure that the problems identified are actually fixed. Software that isn't an application, such as automation created for CI/CD, infrastructure-as-code, and so on, deserves some security testing as well.

[PT3.1: 26] Use external penetration testers to perform deep-dive analysis.

The SSG uses external penetration testers to do a deep-dive analysis on critical software systems or technologies and to introduce fresh thinking. One way to do this is to simulate persistent attackers using goal-oriented red team exercises. These testers are domain experts and specialists who keep the organization up to speed with the latest version of the attacker's perspective and have a track record for breaking the type of software being tested. When attacking the organization's software, these testers demonstrate a creative approach that provides useful knowledge to the people designing, implementing, and hardening new systems. Creating new types of attacks from threat intelligence and abuse cases typically requires extended timelines, which is essential when it comes to new technologies, and prevents checklist-driven approaches that look only for known types of problems.

[PT3.2: 15] Customize penetration testing tools.

Build a capability to create penetration testing tools or adapt publicly available ones to attack the organization's software more efficiently and comprehensively. Creating penetration testing tools requires a deep understanding of attacks (see [AM2.1], [AM3.1]) and technology stacks (see [AM2.2]). Customizing existing tools goes beyond configuration changes and extends tool functionality to find new issues. Tools will improve the efficiency of the penetration testing process without sacrificing the depth of problems that the SSG can identify. Automation can be particularly valuable in organizations using Agile methodologies because it helps teams go faster. Tools that can be tailored are always preferable to generic tools. Success here is often dependent upon both the depth and scope of tests enabled through customized tools.

Deployment: Software Environment (SE)

The Software Environment practice deals with OS and platform patching (including in the cloud), WAFs (web application firewalls), installation and configuration documentation, containerization, orchestration, application monitoring, change management, and code signing.

[SE1.1: 87] Use application input monitoring.

The organization monitors input to the software that it runs in order to spot attacks. Monitoring systems that write log files are useful only if humans or bots periodically review the logs and take action. For web applications, a WAF can do this monitoring, while other kinds of software likely require other approaches, such as runtime instrumentation. Software and technology stacks such as mobile and IoT likely require their own input monitoring solutions. Serverless and containerized software can require interaction with vendor software to get the appropriate logs and monitoring data. Cloud deployments and platform-as-a-service usage can add another level of difficulty to the monitoring, collection, and aggregation approach.



[SE1.2: 115] Ensure host and network security basics are in place.

The organization provides a solid foundation for its software by ensuring that host (whether bare metal or virtual machine) and network security basics are in place across its data centers and networks, and that these basics remain in place during new releases. Host and network security basics must account for evolving network perimeters, increased connectivity and data sharing, software-defined networking, and increasing dependence on vendors (e.g., content delivery, load balancing, and content inspection services). Doing software security before getting host and network security in place is like putting on shoes before putting on socks.

[SE1.3: 79] Implement cloud security controls.

Organizations ensure that cloud security controls are in place and working for both public and private clouds. Industry best practices are a good starting point for local policy and standards to drive controls and configurations. Of course, cloud-based assets often have public-facing services that create an attack surface (e.g., cloud-based storage) that is different from the one in a private data center, so these assets require customized security configuration and administration. In the increasingly software-defined world, the SSG has to help everyone explicitly configure cloud-specific security features and controls (e.g., through cloud provider administration consoles) comparable to those built with cables and physical hardware in private data centers. Detailed knowledge about cloud provider shared responsibility security models is always necessary to ensure that the right cloud security controls remain in place.

[SE2.2: 57] Define secure deployment parameters and configurations.

Create deployment automation or installation guides (e.g., standard operating procedures) to help teams and customers install and configure software securely. Deployment automation usually includes a clearly described configuration for software artifacts and the infrastructure-as-code (e.g., Terraform, CloudFormation, ARM templates, Helm Charts) necessary to deploy them, including details on COTS, open source, vendor, and cloud services components. All deployment automation should be understandable by humans, not just by machines, especially when distributed to customers who buy the software.

[SE2.4: 39] Protect code integrity.

Use code protection mechanisms (e.g., code signing) that allow the organization to attest to the provenance, integrity, and authorization of important code. While legacy and mobile platforms accomplished

this with point-in-time code signing and permissions activity, protecting modern containerized software demands actions in various lifecycle phases. Organizations can use build systems to verify sources and manifests of dependencies, creating their own cryptographic attestation of both. Packaging and deployment systems can sign and verify binary packages, including code, configuration, metadata, code identity, and authorization to release material. In some cases, organizations allow only code from their own registries to execute in certain environments. With many DevOps practices greatly increasing the number of people who can touch the code, organizations should also use permissions and peer review to govern code commits within source code management to help protect integrity.

[SE2.5: 52] Use application containers to support security goals.

The organization uses application containers to support its software security goals. Simply deploying containers isn't sufficient to gain security benefits, while their planned use could support a tighter coupling of applications with their dependencies, immutability, integrity (see [SE2.4]), and some isolation benefits without the overhead of deploying a full operating system on a virtual machine. Containers are a convenient place for security controls to be applied and updated consistently (see [SFD3.2]), and while they are useful in development and test environments, their use in production provides the needed security benefits.

[SE2.7: 42] Use orchestration for containers and virtualized environments.

The organization uses automation to scale service, container, and virtualized environments in a disciplined way. Orchestration processes take advantage of built-in and add-on security features (see [SFD2.1]), such as hardening against drift, secrets management, RBAC, and rollbacks, to ensure that each deployed workload meets predetermined security requirements. Setting security behaviors in aggregate allows for rapid change when the need arises. Orchestration platforms are themselves software that becomes part of your production environment, which in turn requires hardening and security patching and configuration—in other words, if you use Kubernetes, make sure you patch Kubernetes.

[SE3.2: 19] Use code protection.

To protect intellectual property and make exploit development harder, the organization erects barriers to reverse engineering its software (e.g., anti-tamper, debug protection, anti-piracy features, runtime integrity). For some software, obfuscation techniques could be applied as part of the production build and release process. In other cases, these protections could be applied at the software-defined network or software orchestration layer when applications are being dynamically regenerated post-deployment. Code protection is particularly important for widely distributed code, such as mobile applications and JavaScript distributed to browsers. On some platforms, employing Data Execution Prevention (DEP), Safe Structured Handling (SafeSEH), and Address Space Layout Randomization (ASLR) can be a good start at making exploit development more difficult, but be aware that yesterday's protection mechanisms might not hold up to today's attacks.

[SE3.3: 11] Use application behavior monitoring and diagnostics.

The organization monitors production software to look for misbehavior or signs of attack. Go beyond host and network monitoring to look for software-specific problems, such as indications of malicious behavior, fraud, and related issues. Application-level intrusion detection and anomaly detection systems might focus on an application's interaction with the operating system (through system calls) or with the kinds of data that an application consumes, originates, and manipulates. Signs that an application isn't behaving as expected will be specific to the software business logic and its environment, so one-size-fits-all solutions probably won't generate satisfactory results. In some types of environments (e.g., PaaS), some of this data and the associated predictive analytics might come from a vendor.

[SE3.6: 18] Create bills of materials for deployed software.

Create a BOM detailing the components, dependencies, and other metadata for important production software. Use this BOM to help the organization tighten its security posture, that is, to react with agility as attackers and attacks evolve, compliance requirements change, and the number of items to patch grows quite large. Knowing where all the components live in running software—and whether they're in private data centers, in clouds, or sold as box products (see [CMVM2.3])—allows for timely response when unfortunate events occur.



[SE3.8: 0] Perform application composition analysis on code repositories.

Use composition analysis results to augment software asset inventory information with data on all components comprising important applications. Beyond open source (see [SR2.4]), inventory information (see [SM3.1]) includes component and dependency information for internally developed (first-party), commissioned code (second-party), and external (third-party) software, whether that software exists as source code or binary. One common way of documenting this information is to build BOMs. Doing this manually is probably not an option—keeping up with software changes likely requires toolchain integration rather than carrying this out as a point-in-time activity. This information is extremely useful in supply chain security efforts (see [SM3.5]).

Deployment: Configuration Management & Vulnerability Management (CMVM)

The Configuration Management & Vulnerability Management practice concerns itself with operations processes, patching and updating applications, version control, defect tracking and remediation, and incident handling.



[CMVM1.1: 114] Create or interface with incident response.

The SSG is prepared to respond to an event or alert and is regularly included in the incident response process, either by creating its own incident response capability or by regularly interfacing with the organization's existing team. A standing meeting between the SSG and the incident response team keeps information flowing in both directions. Having rebuilt communication channels with critical vendors (e.g., IaaS, SaaS, PaaS) is also very important.

[CMVM1.2: 100] Identify software defects found in operations monitoring and feed them back to engineering.

Defects identified in production through operations monitoring are fed back to development and used to change engineering behavior. Useful sources of production defects include incidents, bug bounty (see [CMVM3.4]), responsible disclosure (see [CMVM3.7]), SIEMs, production logs, and telemetry from cloud security posture monitoring, container configuration monitoring, RASP, and similar products. Entering production defect data into an existing bug-tracking system (perhaps by making use of a special security flag) can close the information loop and make sure that security issues get fixed. In addition, it's important to capture lessons learned from production defects and use these lessons to change the organization's behavior. In the best of cases, processes in the SSDL can be improved based on operations data (see [CMVM3.2]).

[CMVM2.1: 95] Have emergency response.

The organization can make quick code and configuration changes when software (e.g., application, API, microservice, infrastructure) is under attack. An emergency response team works in conjunction with application owners, engineering, operations, and the SSG to study the code and the attack, find a resolution, and fix the production code (e.g., push a patch into production, rollback to a known-good state, deploy a new container). Often, the emergency response team is the engineering team itself. A well-defined process is a must here, but a process that has never been used might not actually work.

[CMVM2.2: 98] Track software defects found in operations through the fix process.

Defects found in operations (see [CMVM1.2]) are entered into established defect management systems and tracked through the fix process. This capability could come in the form of a two-way bridge between defect finders and defect fixers or possibly through intermediaries, but make sure the loop is closed completely. Defects can appear in all types of deployable artifacts, deployment automation, and infrastructure configuration. Setting a security flag in the defect-tracking system can help facilitate tracking.

[CMVM2.3: 62] Develop an operations software inventory.

The organization has a map of its software deployments and related containerization, orchestration, and deployment automation code, along with the respective owners. If a software asset needs to be changed or decommissioned, operations or DevOps teams can reliably identify both the stakeholders and all the places where the change needs to occur. Common components can be noted so that when an error occurs in one application, other applications sharing the same components can be fixed as well. Building an accurate representation of an inventory will likely involve enumerating at least the source code, the open source incorporated both during the build and during dynamic production updates, the orchestration software incorporated into production images, and any service discovery or invocation that occurs in production.

[CMVM3.1: 11] Fix all occurrences of software defects found in operations.

When a security defect is found in operations (see [CMVM1.2]), the organization searches for and fixes all occurrences of the defect, not just the one originally reported. Doing this proactively requires the ability to reexamine the entire operations software inventory

(see [CMVM2.3]) when new kinds of defects come to light. One way to approach reexamination is to create a ruleset that generalizes deployed defects into something that can be scanned for via automated code review. In some environments, fixing a defect might involve removing it from production immediately and making the actual fix in some priority order before redeployment. Use of orchestration can greatly simplify deploying the fix for all occurrences of a software defect (see [SE2.7]).

[CMVM3.2: 19] Enhance the SSDL to prevent software defects found in operations.

Experience from operations leads to changes in the SSDL (see [SM1.1]), which can in turn be strengthened to prevent the reintroduction of defects. To make this process systematic, incident response postmortem includes a feedback-to-SSDL step. The outcomes of the postmortem might result in changes such as to tool-based policy rulesets in a CI/CD pipeline and adjustments to automated deployment configuration (see [SE2.2]). This works best when root-cause analysis pinpoints where in the software lifecycle an error could have been introduced or slipped by uncaught (e.g., a defect escape). DevOps engineers might have an easier time with this because all the players are likely involved in the discussion and the solution. An ad hoc approach to SSDL improvement isn't sufficient for prevention.

[CMVM3.3: 18] Simulate software crises.

The SSG simulates high-impact software security crises to ensure that software incident detection and response capabilities minimize damage. Simulations could test for the ability to identify and mitigate specific threats or, in other cases, begin with the assumption that a critical system or service is already compromised and evaluate the organization's ability to respond. Planned chaos engineering can be effective at triggering unexpected conditions during simulations. The exercises must include attacks or other software security crises at the appropriate software layer to generate useful results (e.g., at the application layer for web applications and at lower layers for IoT devices). When simulations model successful attacks, an important question to consider is the time required to clean up. Regardless, simulations must focus on security-relevant software failure, not on natural disasters or other types of emergency response drills. Organizations that are highly dependent on vendor infrastructure (e.g., cloud service providers, SaaS, PaaS) and security features will naturally include those things in crisis simulations.

[CMVM3.4: 26] Operate a bug bounty program.

The organization solicits vulnerability reports from external researchers and pays a bounty for each verified and accepted vulnerability received. Payouts typically follow a sliding scale linked to multiple factors, such as vulnerability type (e.g., remote code execution is worth \$10,000 vs. CSRF is worth \$750), exploitability (demonstrable exploits command much higher payouts), or specific service and software versions (widely deployed or critical services warrant higher payouts). Ad hoc or short-duration activities, such as capture-the-flag contests or informal crowdsourced efforts, don't constitute a bug bounty program.

[CMVM3.5: 13] Automate verification of operational infrastructure security.

The SSG works with engineering teams to verify with automation the security properties (e.g., adherence to agreed-upon security hardening)

of infrastructure generated from controlled self-service processes. Engineers use self-service processes to create networks, storage, containers, and machine instances, to orchestrate deployments and to perform other tasks that were once IT's sole responsibility. In facilitating verification, the organization uses machine-readable policies and configuration standards (see [SE2.2]) to automatically detect issues and report on infrastructure that does not meet expectations. In some cases, the automation makes changes to running environments to bring them into compliance, but in many cases, organizations use a single policy to manage automation in different environments, such as in multi- and hybrid-cloud environments.

[CMVM3.6: 3] Publish risk data for deployable artifacts.

The organization collects and publishes risk information about the applications, services, APIs, containers, and other software it deploys. Whether captured through manual processes or telemetry automation, published information extends beyond basic software security (see [SM2.1]) and inventory data (see [CMVM2.3]) to include risk information. This information usually includes constituency of the software (e.g., BOMs, [SE3.6]), what group created it and how, and the risks associated with known vulnerabilities, deployment models, security controls, or other security characteristics intrinsic to each artifact. This approach stimulates cross-functional coordination and helps stakeholders take informed risk management action. Making a list of risks that aren't used for decision support won't achieve useful results.

[CMVM3.7: 20] Streamline incoming responsible vulnerability disclosure.

Provide external bug reporters with a line of communication to internal security experts through a low-friction, public entry point. These experts work with bug reporters to invoke any necessary organizational responses and to coordinate with the external entities throughout the defect management lifecycle. Successful disclosure processes require insight from internal stakeholders such as legal, marketing, and public relations roles to simplify and expedite decision-making during software security crises. Although bug bounties might be important to motivate some researchers (see [CMVM3.4]), proper public attribution and a low-friction reporting process is often sufficient motivation for researchers to participate in a coordinated disclosure. Most organizations will use a combination of easy-to-find landing pages, common email addresses (security@), and embedded product documentation when appropriate (security.txt) as an entry point for external reporters to invoke this process.



[CMVM3.8: 0] Do attack surface management for deployed applications.

Operations standards and procedures proactively minimize application attack surfaces by using attack intelligence and application weakness data to limit vulnerable conditions. Finding and fixing software bugs in operations is important (see [CMVM1.2]) but so is finding and fixing errors in cloud security models, VPNs, segmentation, security configurations for networks, hosts, and applications, and so on to limit the ability to successfully attack deployed applications. Combining attack intelligence (see [AM1.5]) with information about software assets (see [AM3.3]) and a continuous view of application weaknesses helps ensure that attack surface management keeps pace with attacker methods. SBOMs (see [SE3.6]) are also an important information source when doing attack surface management in a crisis.

APPENDICES

A. ROLES IN A SOFTWARE SECURITY INITIATIVE

An SSI requires thoughtful staffing with both full-time and dotted-line people. You can use the descriptions below to help define roles and responsibilities that accommodate your needs for execution and growth.

In Part 4 of this report, we provided a summary of the different roles involved in implementation of the SSI. Here, we provide details and data about those roles.

EXECUTIVE LEADERSHIP

Historically, security initiatives that achieve firm-wide impact are sponsored by a senior executive who creates an SSG where software security governance and testing are distinctly separate from software delivery. Security initiatives without that executive sponsorship, by comparison, have historically had little lasting impact across the firm. By identifying a senior executive and putting them in charge of software security, the organization can address two “Management 101” concerns: accountability and empowerment.

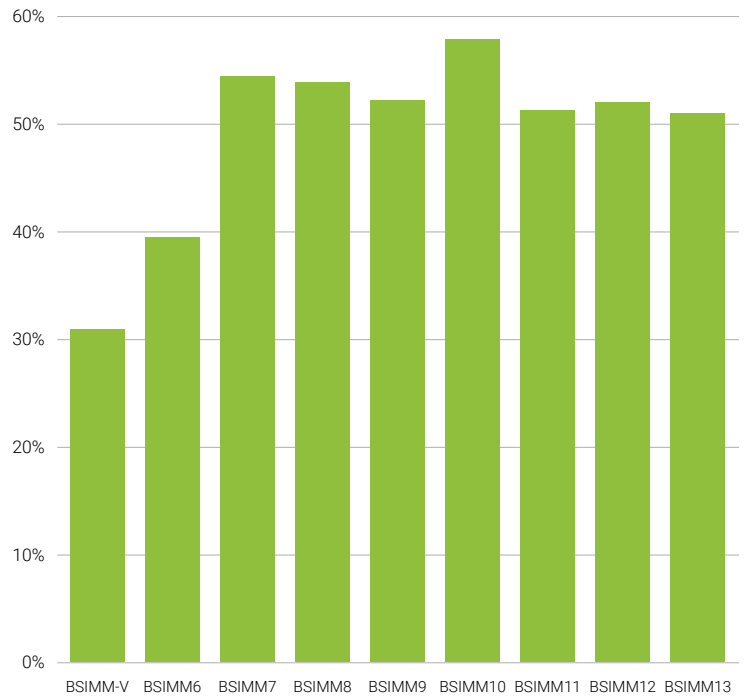


FIGURE 5. PERCENTAGE OF SSGS WITH A CISO AS THEIR NEAREST EXECUTIVE. Assuming new CISOs generally receive responsibilities for SSIs, this data suggests that CISO role creation is also flattening out.

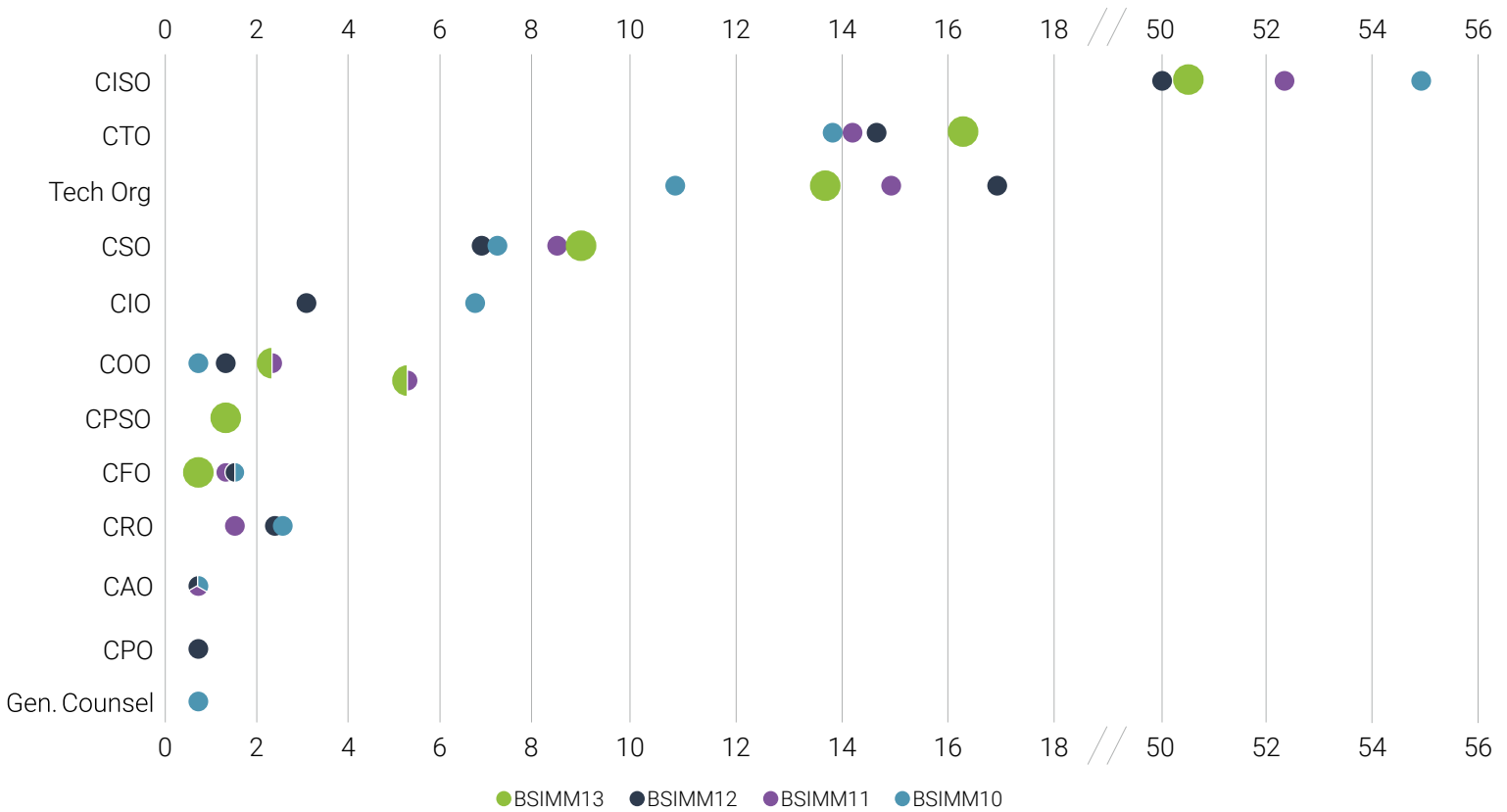


FIGURE 6. NEAREST EXECUTIVE TO SSG. Although many SSGs have a CISO as their nearest executive, we see a variety of executives overseeing software security efforts in the 130 BSIMM13 firms.

In BSIMM-V, we saw CISOs as the nearest executive in 21 of 67 firms, which grew in BSIMM6 to 31 of 78, and again for BSIMM7 with 52 of 95. Since then, the percentage has remained relatively flat even as the BSIMM community has grown, as shown in Figure 5.

If we look across all the executives nearest to SSG owners, not just CISOs, we observe a large spread in the reporting path to executive leadership for BSIMM10 through BSIMM13, as shown in Figure 6. The larger green circles show by percentage the SSG leader's nearest executive in the BSIMM13 data pool, while smaller circles show the percentages for previous BSIMMs. For example, a CISO is the closest executive in 51% of organizations in the BSIMM13 community, and that percentage ranged from 50% to 55% in BSIMM10 through BSIMM12. For the BSIMM13 data pool, we no longer see SSGs reporting to CRO (risk), CAO (assurance), CPO (privacy), and General Counsel roles. Note that for BSIMM13, we added 27 firms and removed 25, which also affects analysis of reporting chains. Of course, not all people with the same title perform, prioritize, enforce, or otherwise provide resources for the same efforts the same way across various organizations.

CISOs in turn report to different executives among the 130 BSIMM13 firms. Figure 7 shows that CISOs report most commonly to CIOs (26 of 66, or almost 40% of the time) and report directly to the CEO less than 11% of the time (7 of 66).

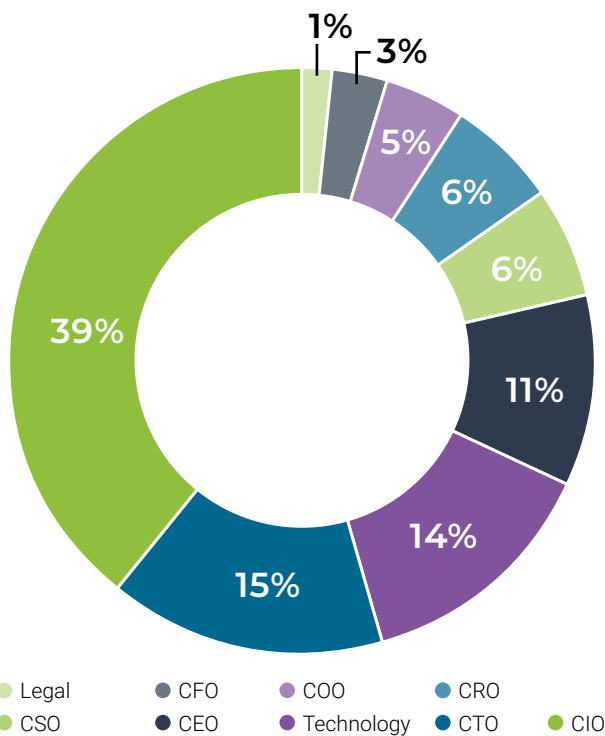


FIGURE 7. TO WHOM THE CISO REPORTS. In the BSIMM13 community, the CISO reports to a variety of roles, with the most common being the CIO, CTO, and a technology executive (e.g., head of engineering, architecture, or software).

SOFTWARE SECURITY GROUP LEADERS

SSG leaders are individuals in charge of day-to-day efforts in the 130 SSGs we studied for BSIMM13. They have a variety of titles, such as the following:

- Application Security Architect
- Application Security Manager
- Director Application Security
- Director Cybersecurity
- Director IT Risk Management
- Director IT Shared Services
- Director Product Security
- Director Security Assurance
- Executive Director Product Security
- Information Assurance Director
- Lead Security Architect
- Manager Software Security Engineering
- Product Security AppSec
- Security Architect
- Security Director
- Security Engineering Manager
- Senior Director Product Security
- SVP Product Security & Technology
- VP Product and Application Security
- VP Security Architecture
- VP Security Compliance

As shown in Figure 8, SSG leaders are typically one or two hops from their nearest executive (e.g., a CxO or related technology organization title). In addition, we observed that this nearest executive is usually a further one or two hops away from the CEO. When the SSG leader is an executive themselves, which happens 10% of the time (13 out of 130), they are CISOs almost 70% of the time (9 out of 13), with other titles being CTO, CPSO (Chief Product Security Officer), and CSO.

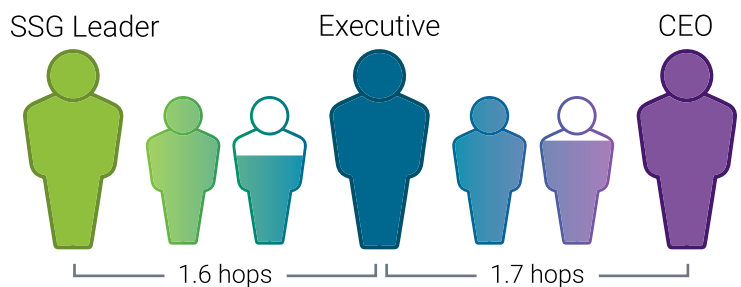


FIGURE 8. SSG LEADERSHIP REPORTING CHAINS. SSG leaders are typically three or four hops away from the CEO. When SSG leaders are themselves an executive, they most often have a security role—CISO, CPSO (Chief Product Security Officer), CSO—but some have a technology role such as CTO.

SOFTWARE SECURITY GROUP (SSG)

Each of the 130 initiatives in BSIMM13 has an SSG—an organizational group dedicated to software security. In fact, without an SSG, successfully carrying out BSIMM activities across a software portfolio is very unlikely, so the creation of an SSG is a crucial first step. The SSG might start as a team of one—just the SSG leader—and expand over time. The SSG might be entirely a corporate team, entirely an engineering team, or an appropriate hybrid. The team’s name might also have an appropriate organizational focus, such as application security group or product security group, or perhaps DevSecOps.

Some SSGs are highly distributed across a firm whereas others are centralized. Even within the most distributed organizations, we find that software security activities are almost always coordinated by an SSG.

Although no two of the 130 firms we examined had exactly the same SSG structure, we did observe some commonalities. At the highest level, SSGs seem to come in five overlapping structures:

- Organized to provide software security services
- Organized around setting and verifying adherence to policy
- Designed to mirror business unit organizations
- Organized with a hybrid policy and services approach
- Structured around managing a matrixed team of experts doing software security work across the development or engineering organizations.

Table 4 shows some SSG-related statistics across the 130 BSIMM13 firms, but note that a handful of large outliers affect the numbers this year. The “Notes - No Outliers” column shows the effect of removing

outliers, or the top 10 firms, for that SSG characteristic. Refer to Appendix H for more details on how SSGs evolve over time.

SATELLITE (SECURITY CHAMPIONS)

In addition to the SSG, many SSIs have identified individuals (often developers, testers, architects, and cloud and DevOps engineers) who are a driving force in improving software security but are not directly employed in the SSG. We collectively refer to them as the satellite, and many organizations refer to this group as their software security champions. A satellite can enable an SSI to scale its efforts while reducing dependency on the SSG team, and there appears to be a correlation between a higher BSIMM score and the presence of satellite, as shown in Figure 9. Having satellite members carry out software security activities empowers engineering teams to own their software security deliverables and removes SSG members from the engineering critical path.

Satellite members are often chosen for software portfolio coverage (with one or two members in each engineering group), and sometimes for reasons such as technology stack coverage or geographical reach. The satellite can act as a sounding board for the feasibility and practicality of proposed software security changes and improvements. Understanding how SSI governance changes might affect project timelines and budgets helps the SSG proactively identify potential frictions and minimize them.

Successful satellite groups get together regularly to compare notes, learn new technologies, and expand stakeholder understanding of the organization’s software security challenges. Motivated individuals often share digital work products, such as sensors, code, scripts,

THE SOFTWARE SECURITY GROUP					
STATISTICS	AVERAGE	MEDIAN	LARGEST	SMALLEST	NOTES – NO OUTLIERS
SSG Size	25.7	8	892	1	Average drops to 19.0 (1 outlier)
SSG Member to Developer Ratio	3.01%	0.88%	51.43%	0.06%	Average drops to 2.63% (1 outlier) Average drops to 1.45% (no top 10)
SSG Member to Developer Ratio (800+ developers) – 64 firms	1.03%	0.56%	14.86%	0.06%	800 – median number of dev
SSG Member to Developer Ratio (less than 800 developers) – 66 firms	5.11%	1.79%	51.43%	0.33%	
Number of Developers	3,146	800	100,000	25	Average drops to 2,395 (1 outlier) Average drops to 1,362 (no top 10)
Number of Applications	1,118	159	40,000	1	Average drops to 816 (1 outlier) Average drops to 475 (no top 10)
SSG Age	5.0	4.0	23.0	0.1	
Satellite to Developer Ratio	4.20%	1.24%	57.14%	0%	Average drops to 3.79% (1 outlier) Average drops to 2.44% (no top 10)
Satellite to Developer Ratio (800+ developers) – 64 Firms	3.19%	1.50%	25.63%	0%	
Satellite to Developer Ratio (less than 800 developers) – 66 Firms	5.26%	1.20%	57.14%	0%	
SSG to Application Ratio	40.64%	6%	1100%	0.01%	Average drops to 14.80% (2 outliers) Average drops to 9.6% (no top 10)

TABLE 4. THE SOFTWARE SECURITY GROUP. We calculated the ratio of full-time SSG members to developers for the entire data pool by averaging the individual ratio for of each participating firm. When planning the size and structure of your own SSG, consider the number of developers and applications to determine what resources you need to scale the SSI. In the Notes column, we show the impact of removing outliers in the data.

tools, and security features, rather than, for example, getting together to discuss enacting a new policy. Specifically, these proactive champions are working bottom-up and delivering software security features and awareness through implementation.

For more information about security champions, refer to Appendix G.

KEY STAKEHOLDERS

SSIs are truly cross-departmental efforts that involve a variety of stakeholders:

- Builders, including developers, architects, and their managers, must practice security engineering, taking some responsibility for both the definition of “secure enough” as well as ensuring that what’s delivered achieves the desired posture. An SSI requires collaboration between the SSG and these engineering teams to carry out the activities described in the BSIMM.
- Testers typically conduct functional and feature testing, but moving on to include security testing is very useful. Some testers are beginning to anticipate how software architectures and infrastructures can be attacked and are working to find an appropriate balance between automated and manual testing to ensure adequate security testing coverage.
- Operations teams must continue to design, defend, and maintain resilient environments because software security doesn’t end when software is “shipped.” In accelerating trends, development and operations are collapsing into one or more DevOps teams, and the business functionality delivered is becoming very dynamic. This means that an increasing amount of security effort, including infrastructure controls and security configuration, is becoming software defined (and that software should also be secure).
- Administrators must understand the distributed nature of modern systems, create and maintain secure configurations, and practice the principle of least privilege, especially when it comes to host, network, infrastructure, and cloud services for deployed applications.
- Executives and middle management, including business owners and product managers, must understand how early investment in security design and analysis affects the degree to which users will trust their products. Business requirements should explicitly address security needs, including security-related compliance. Any sizable business today depends on software to work; thus, software security is a business necessity. Executives are also the group that must provide resources for new efforts that directly improve software security and must actively support digital transformation efforts related to infrastructure- and governance-as-code.
- Data privacy specialists form an integral part of the software security effort in some firms, combining forces with security specialists when engaging with engineering. They might be responsible for analysis of privacy regulations, definition of privacy requirements, and tracking of PII and other data categories. This has become increasingly common in response to regulations such as GDPR.
- Vendors, including those who supply on-premises products, custom software, and SaaS, are increasingly subjected to SLAs and reviews (such as the Payment Card Industry [PCI] Software Security Framework [SSF] and the BSIMMsc) to help ensure that their products are the result of an SSDL. Of course, not all software (e.g., open source) comes from a vendor.

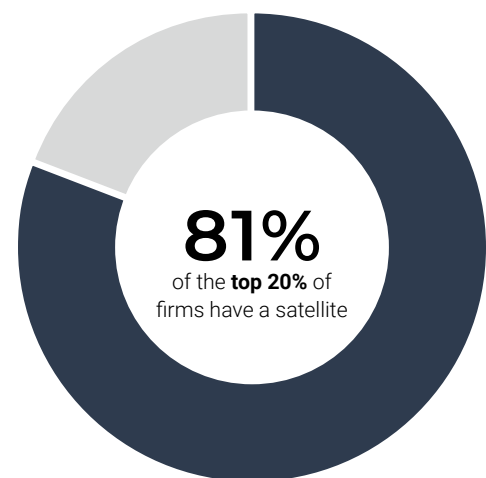
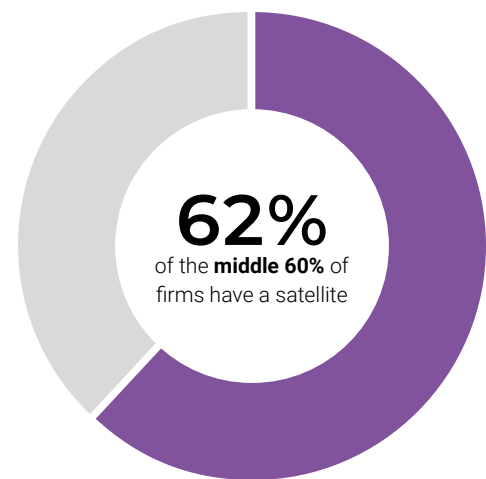
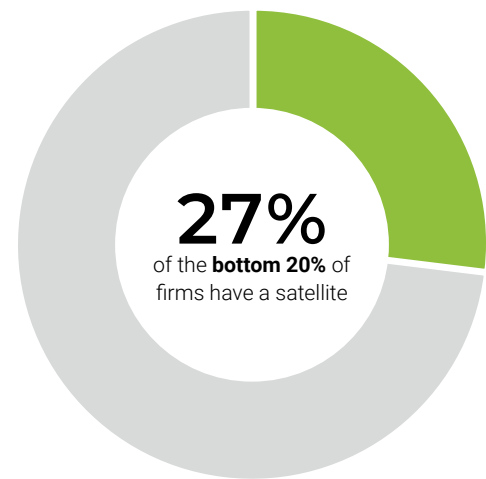


FIGURE 9. THE SATELLITE AND THE BSIMM SCORE. Eighty-one percent of the top-scoring firms in the BSIMM13 community have a satellite (security champions). In contrast, fewer than 30% of bottom-scoring firms have one.

B. HOW TO BUILD OR UPGRADE AN SSI

- Putting someone in charge is just a first step in building an SSI. There will be iterations of planning, growth, measurement, and bridge-building. You can use the processes below to guide your SSI's growth
- from newly emerging through dependable maturity.

The BSIMM is not just a long-term software security study, or a single-purpose SSI benchmarking tool—it also eases management and evolution for anyone in charge of software security, whether that person is currently in a central governance-focused position or in a more local engineering-focused team. Firms of all maturity levels, sizes, and verticals use the BSIMM as a reference guide when building new SSIs and when evolving their initiatives through various maturity and stakeholder ownership phases over time.

We often refer to SSIs we've seen as being in one of three broad states—emerging, maturing, and enabling—which we describe as follows:

- **Emerging.** An emerging SSI has defined its initial strategy, has chosen foundational activities (e.g., such as those observed most frequently in the community), has acquired some resources, and has a general roadmap for the next 18 months. SSI leaders are likely resource-constrained on both people and budget, so the SSG is usually small and uses compliance requirements or other executive mandates to drive participation and to continue adding activities. These leaders require strong, visible, and ongoing executive support to manage frictions with key stakeholders who are resistant to adopting foundational process discipline.
- **Maturing.** A maturing SSI has an in-place team, defined processes for interacting with software security stakeholders, and a documented software security approach that is clearly connected to executive expectations for both managing software security risk and progressing along a roadmap to scale security capabilities. A maturing SSI is learning from its existing efforts, likely making consistent, incremental improvements in the SSDL and key security integrations. Example improvements include:
 - Reducing friction across business and development stakeholders
 - Protecting people's productivity gains through automation investments
 - Building bridges to other parts of the firm through evangelism, defect discovery, software supply chain protection, and incident response
 - Undergoing a "shift everywhere" transformation to test software artifacts as soon as appropriate
 - Adjusting the security strategy to keep pace with changes in risk and risk management processes
 - Finding solutions to systemic problems and making them broadly available as reusable, pre-approved IP
 - Responding quickly when attacks or other circumstances uncover a lack of resiliency
- **Enabling.** An enabling SSI ensures that all stakeholders can meet their objectives without putting the organization at unacceptable risk. The following are important principles for an enabling SSI:

- There is a continuous evangelizing about the best way for all stakeholders to meet security expectations, ensuring that the path of least resistance for development and deployment is also the most secure path, and investing to proactively overcome various people, process, technology, and cultural growing pains.
- The evolutionary needs of the SSI are harmonized with the goals of business initiatives, such as digital transformation, open source use, and cloud adoption.
- A mature and integrated response to process and technical risk invokes an innovation engine to make reasonably future-proof solutions.
- The use of culturally engrained approaches to automation, blameless review of failures, and protecting critical resources—people, for example—allow more time to tackle security innovation.
- A platform-engineering perspective removes security activity silos and ensures that all telemetry and benefits are available to all stakeholders everywhere.

It's compelling to imagine that organizations could reach a state of emerging, maturing, or enabling simply by applying a certain number or mix of activities to specific percentages of the staff and software portfolio, but that doesn't happen. Experience shows that SSIs usually reach an emerging stage by organizing all the ad hoc software security efforts they're already doing into one program. The SSIs usually proceed to the maturing stage by focusing on the activities that are right for them without regard for the total activity count. This is especially true when considering the complexity of scaling some activities across 100, 1,000, or 10,000+ applications or people.

Organizations rarely move their entire SSI from emerging to enabling all at once. We have seen SSIs form, break up, and re-form over time, so an SSI might shift between emerging, maturing, and enabling a few times over the years. In addition, capabilities within an SSI (e.g., supply chain security, training) likely won't progress through the same states at the same rate. We've noted cases where one capability—vendor management, for example—might be emerging, while the defect management capability is maturing, and the defect discovery capability is in an enabling stage. There is also constant change in tools, skill levels, external expectations, attackers, attacks, resources, culture, and everything else. You can use the BSIMM13 community scorecard (see Figure 17) to see the frequency with which the BSIMM activities are observed across all participants, but use your own metrics to determine if you're making the progress that's right for you.

STARTING AN SSI: GETTING TO AN EMERGING STATE

It's unlikely that any organization is doing *nothing* about software security. Even an organization without a formal initiative or a defined owner likely has some software security policy, application security testing, and processes for working with stakeholders. Provided below are actionable steps for consolidating that ad hoc effort into an emerging SSI. Keep in mind that most SSIs are multiyear efforts with real budget, mandate, and ownership behind them. In addition, while all initiatives look different and are tailored to fit a particular organization, most initiatives share common core activities (see Table 7).

Figure 10 organizes the steps and suggested timeline to establish an emerging SSI, along with the associated BSIMM activities. It also includes a notional level of effort anticipated across people and budget, as well as estimated duration, all on a 1 – 3 scale. The effort and cost to reach each of these goals will vary across companies, of course, but is primarily affected by risk objectives, organizational structure, and portfolio size. For example, deploying on-site static analysis across 10 applications using a common pipeline in one business unit will likely have a lower level of effort than deploying that static analysis across 10 applications built in 10 toolchains in 10 business units.

Note that the getting started roadmap shown in Figure 10 includes some activities that have a high impact for emerging SSIs even though they appear to be rarely observed in the BSIMM community. This happens because newly added BSIMM activities start with an observation rate of zero (e.g., [ST3.6] added for BSIMM11). These are foundational activities, even if organizations are just starting to add them to their journeys. Importantly, the steps described here are not specific to where in the organization the SSG is created. The SSG can be centralized in a governance group or an engineering group, or it can be decentralized across both. Regardless, governance and engineering functions will have to cooperate to ensure the achievement of organizational software security goals.

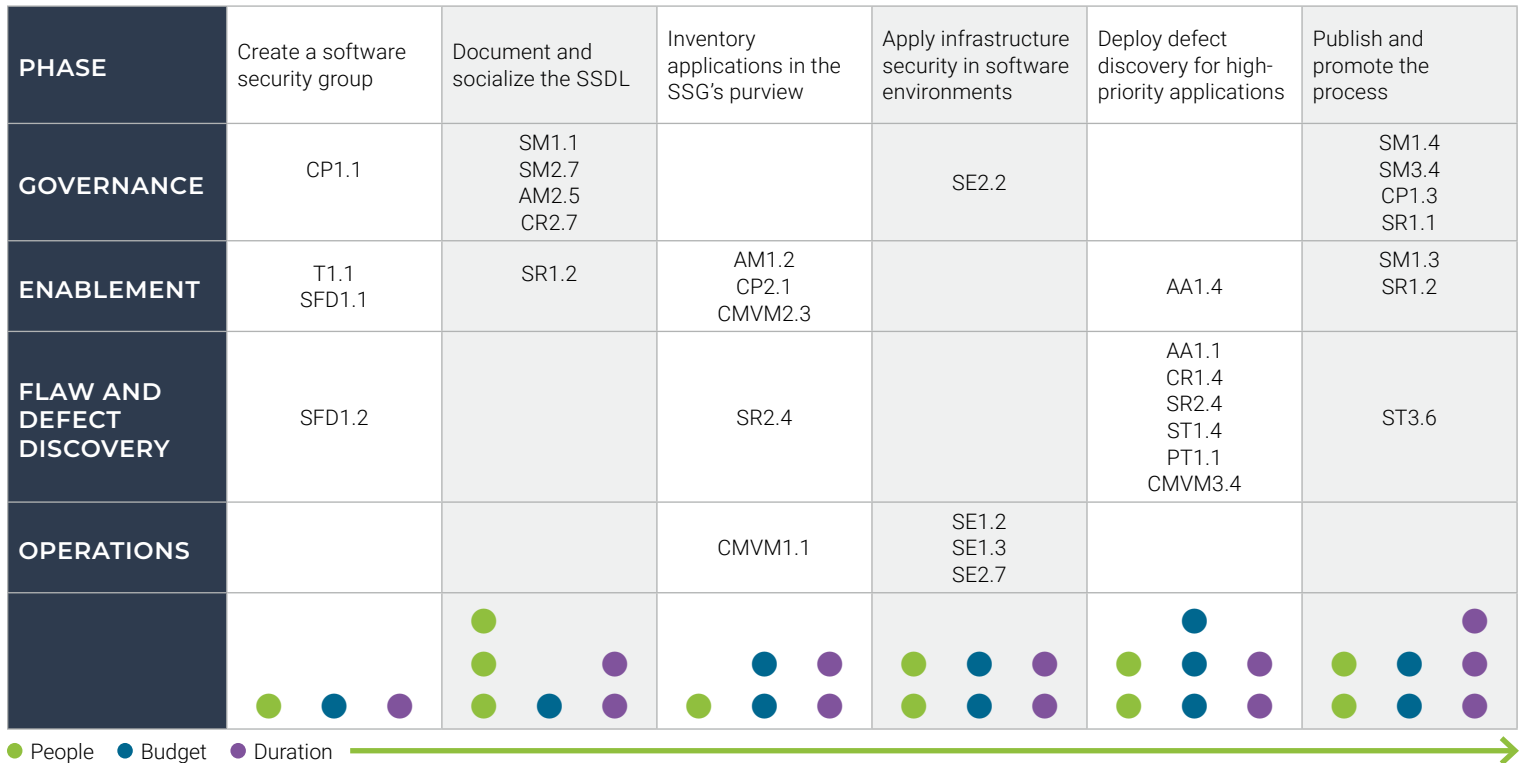
Note that an SSG leader with a young initiative (e.g., less than 18 months) working on foundations should not expect or set out to quickly implement too many BSIMM activities. Firms can absorb only a limited amount of technology, hiring, cultural, and process change

at any given time. The BSIMM13 data shows that SSIs having an age of 18 months or less at the time of assessment (28 of 130 firms) have an average score of 32.9.

Following are some details on the steps shown in Figure 10, and these steps include activity references. The references are meant to help the reader understand the associations between the topic being discussed and one or more BSIMM activities. Note that the references don't mean the topic being discussed is fully equivalent to the activity. For example, when we say, "...initial inventory [AM1.2]" (i.e., *Use a data classification scheme for software inventory*), we don't mean that having the initial inventory encompasses the totality of [AM1.2], just that having it will likely be something you'll do on your way to implementing [AM1.2]. To continue using [AM1.2] as an example, most organizations will not set about implementing this activity and get it all done all at once. Instead, an organization will likely create an initial classification scheme and inventory, implement a process to keep the inventory up to date, and then decide how to create a view that's meaningful for stakeholders. Every activity has its own nuances and components, and every organizational evolution path for its emerging SSI will be unique.

Create a Software Security Group

The most important first step for all SSIs is to have a dedicated SSG that can get resources and drive organizational change, even if it's a group of one person coordinating organizational efforts. The SSG must then understand which software security goals are important to the business and establish policy and process to drive everyone in that direction. At a minimum, the SSG should identify the



The arrow of time (x-axis) is a notional order of efforts. Although this diagram appears to depict a waterfall process, many of these efforts will be happening at the same time and some will be repeated multiple times.

FIGURE 10. GETTING STARTED ROADMAP WITH NOTIONAL EFFORTS. This roadmap is supplemented with relative effort levels so that organizations can plan the resources needed for their emerging SSI.

risk management, compliance, and contractual requirements that the organization must adhere to [CP1.1]. Using awareness training [T1.1] to then help ensure that everyone understands their security responsibility is a common approach.

The SSG must work with engineering teams to establish a common understanding of the approach to software security. The approach might be to set up automated defect discovery, address security questions from developers with reusable security features [SFD1.1], and act as an advisor for design decisions [SFD1.2].

Document and Socialize the SSDL

Publish security policies and standards through established governance, risk, and compliance (GRC) channels to complement existing IT security standards or create those channels as necessary. The SSG can also create a security portal (e.g., website, wiki) that houses SSDL information centrally [SR1.2]. Similar to the approach for prioritizing defect discovery efforts by categorizing attacks and bugs [AM2.5, CR2.7], we observe these firms driving initial standards creation from industry top risks, leveraging sources such as MITRE, ISO, and NIST to form baseline requirements.

Getting the word out about the organization's top risks and what can be done about them is a key part of the SSG's job. We observe these leaders using every channel possible (e.g., town halls, brown bags, communities of practice forums, messaging channels) to socialize the software security message and raise awareness of the SSDL [SM2.7].

Inventory Applications in the SSG's Purview

One of the first activities for any SSG is to create an initial inventory of the application portfolio under its purview [AM1.2, CMVM2.3]. As a starting point, the inventory should include each application's important characteristics (e.g., programming language, architecture type, open source used [SR2.4]). Particularly useful for monitoring

and incident response activities [CMVM1.1], many organizations will include relevant operational data in the inventory (e.g., where the application is deployed, owners, emergency contacts).

Inventory efforts tend to favor a top-down approach in the beginning, usually starting with a questionnaire to elicit data from business managers who serve as application owners, then using tools to find open source software. The SSG also tends to focus on understanding where sensitive data resides and flows (e.g., PII inventory) [CP2.1] and the resulting business risk level associated with the application (e.g., critical, high, medium, low).

When working with engineering teams, these efforts commonly attempt to extract software inventory data from the tools used to manage IT assets. By scraping these software and infrastructure configuration management databases (CMDBs) or code repositories, the SSG crafts an inventory brick by brick rather than top-down.

Maintaining an application inventory is a capability to be built over time rather than a one-time effort. To remain accurate and current, the inventory must be regularly monitored and updated. As with all data currency efforts, it's important to make sure the data isn't overly burdensome to collect and is periodically spot-checked for validity. Organizations should favor automation for application discovery and management whenever possible.

Apply Infrastructure Security in Software Environments

Bad infrastructure security can undermine good software security, which means the SSG must ensure host and network security basics are in place [SE1.2] as well as cloud security controls [SE1.3]. Security engineers might begin by conducting this work manually, then baking these settings and changes into their software-defined infrastructure scripts [SE2.2] to ensure both consistent use within a development team and scalable sharing across the organization.

Forward-looking organizations that have adopted software and network orchestration technologies [SE2.7] (e.g., Kubernetes, Envoy, Istio) get maximum impact from them with the efforts of even an individual contributor, such as a security-minded DevOps engineer. Though many of the technologies in which security engineers specify hardening and security settings are human-readable, engineering groups don't typically take the time to extract and distill a document-based security policy from these codebases.

Deploy Defect Discovery for High-Priority Applications

Regardless of business drivers, one of the quickest ways of transitioning unknown risk to managed risk is through defect discovery. Use automated tools, both static and dynamic, to provide fast, regular insight into the portfolio security posture, with experts doing detailed testing for important applications [AA1.1, CMVM3.4]. While not necessarily done for the entire application portfolio, conducting some targeted vulnerability discovery to get a feel for the current risk posture allows firms to motivate the necessary conversations with stakeholders to gain buy-in and prioritize remediation. Organizations tend to determine their high-priority applications via risk ranking [AA1.4]. Phase in a combination of manual testing techniques against these high-priority applications and rely on automated testing techniques for portfolio coverage.

CHECKLIST FOR EMERGING SSGs

- 1. Create an SSG.** Put a dedicated group in charge and give them resources.
- 2. Document and Socialize the SSDL.** Tell all stakeholders the expectations for software security.
- 3. Inventory Applications.** Decide on what you're going to focus on first, then apply good risk management.
- 4. Apply Infrastructure Security.** Don't put good software on bad systems or in poorly constructed networks (cloud or otherwise).
- 5. Deploy Defect Discovery.** Determine the issues in today's in-progress and production applications, then plan for tomorrow.
- 6. Publish and Promote.** Roll out the secure SDLC and promote it both bottom-up and top-down.
- 7. Progress to the Next Step.** Pick new activities to focus on, whether they are emerging or moving to the maturing stage.

Static and dynamic software testing techniques each provide unique views into an application's security posture. Static analysis can look for issues inside the code the organization develops [CR1.4] and inside third-party components [SR2.4]. Dynamic application security tests [ST1.4] can uncover immediately exploitable issues and help provide steps to reproduce attacks. In addition, QA groups can help ensure that development streams are adhering to security expectations. All these testing results assist with prioritization and displaying impact to executive leadership.

Manual testing efforts generally start by bringing in third-party assessors [PT1.1] on a regular cadence, either upon major milestones or, more commonly, as a periodic out-of-band exercise to assess the most critical applications. Even where an internal penetration testing function exists, a third party periodically bringing in a unique perspective will be beneficial.

Note that engineering groups will tend to favor empowering pipelines and testers with automation and allow engineering leadership or individual engineering teams to define some aspects of mandatory testing and remediation timelines. It's important to ensure static, dynamic, and manual testing creates minimal unnecessary friction in engineering processes.

Publish and Promote the Process

With a strategy in hand, an understanding of the portfolio, and security expectations set with engineering teams, the SSG documents the SSDL [SM1.1] and begins collecting telemetry [SM1.4]. The SSDL should include clearly documented goals, roles, responsibilities, and activities. The most usable SSDLs include process diagrams and provide contextual details for each stakeholder. Many organizations seeking to consolidate ad hoc efforts into an emerging SSI will find a variety of SSDLs in use across engineering teams. In these cases, the new SSDL might be a replacement for all such approaches, but it might also have some parts that are abstract enough to account for all processes until they can be rolled into the new approach. Publication of this process is also a good time for the SSG to start a software security hub where the SSG can disseminate knowledge about the process and about software security as a whole [SR1.2].

In a top-down approach, organizations favor creating policy [CP1.3] and standards [SR1.1] that can be followed and audited like any other business process. Rather than documents, however, engineering teams might favor implementing their part of an SSDL inside of pipelines [SM3.4] and scripts [ST3.6], or by prescribing reusable security blocks that meet expectations. Over time, the SSG will also have to deliver some policy in the form of governance-as-code in engineering pipelines [SM1.4].

While executives have likely been engaged to get the SSI to this point, this is a good time to ensure that they're being regularly kept up to date with software security. Remember, executive teams need to understand not only how the SSI is performing but also how other firms are solving software security problems and the ramifications of not investing in software security [SM1.3].

Progress to the Next Step in Your Journey

Usually done as part of moving to the mature stage, the SSG then proceeds to scale the SSI. For example, this scaling might be done through creating a champions program, improving the inventory capability based on lessons learned, automating the basics, doing more prevention, and then repeating. As the initiative matures and the business grows, there will be new challenges for the SSG to address, so it will be crucial to ensure that feedback loops are in place for the program to consistently measure its progress and maturity.

LESSONS FROM THE COMMUNITY

The purpose of the BSIMM is to measure SSIs. While the BSIMM doesn't directly measure SSI architecture, evolution, or motivations, our experience with over 254 organizations since 2008 has highlighted cultural differences in SSI implementations.

No SSI is built in a vacuum. Whether your SSI is just emerging or has some capabilities in the maturing stage, knowledge from both the struggles and successes of other organizations can save you time and disruption. As software security becomes an important goal for any organization, multiple internal groups might each be taking their own approach to their goals. Understanding and harmonizing these cultural and technological views into a single SSI is important to long-term success.

Whether your SSI is just emerging or has some capabilities in the maturing stage, knowledge from both the struggles and successes of other organizations can save you time and disruption.

Cultures

Whether implicitly or explicitly, organizations choose the path for their software security journey by tailoring goals, methods, tools, resources, and approaches according to their individual cultures. There have always been two distinct cultures in the BSIMM community:

- Organizations where the SSG was started by executives in a central corporate group (e.g., under a CISO) as a full-time role and chartered with software security governance, including compliance, testing, remediation monitoring, and risk management. This SSG stayed in the corporate organization chart, had the power to enact organization-wide policy, and expanded its efforts outward through, for example, tooling and security champions. This path was seen most often in regulated industries such as banking, insurance, FinTech, and healthcare but was also seen in ISV and technology firms.

- Organizations where the SSG was started by engineering technical leadership (e.g., senior application architects) as a part-time role and focused on technical software security efforts, such as configuration hardening, technology stack standards, secure coding standards, and security tool integration, which was often done for a single tool chain or project. As evangelism efforts convinced other development projects to use the same technical controls, the technical leadership usually worked with a CTO, VP Engineering, or other technology executive to establish a centralized security function within the engineering domain. The centralized function—often still part time—then used its influence to establish its own type of governance, which was often peer pressure to set some development process, create and manage security standards, and ensure that the silos of engineering, testing, and operations were aware of and adhered to general security expectations. This path was most often seen in technology, cloud, and ISV firms but was also seen in other verticals.

Regardless of its origin point, both cultures usually arrived at an SSI that is driven by a centralized, dedicated SSG whose function is to ensure that appropriate software security activities are happening across the portfolio. That is, nearly all SSIs that are more than a couple years old are driven top-down by governance objectives, even those started by engineering for engineering. Evangelism, peer pressure, and local implementations go only so far in formally implementing software security risk management as a culture.

Today, as you start or plan a major revamp of your SSI, just get started. You can start in corporate, or you can start in engineering. You can start with governance as a top priority, or you can focus on some technical controls. In any case, history seems to show that SSIs gravitate toward a focus on policy along with process that ensures adherence. Yours likely will as well.

A New Wave in Engineering Culture

Over the past few years, we’re seeing a new wave of software security efforts emerging from engineering teams. These teams are usually responsible for delivering a product or value stream—such as is common within ISVS—or maintaining a technology domain—such as the “cloud security group” or a part of some digital transformation group. Some organizations refer to these collective security efforts as site reliability engineering, DevSecOps, or GitOps security, and some have no specific name for it at all.

At least three factors are driving these new efforts:

- The confluence of process friction, unpredictable impacts on delivery schedules, adversarial internal relationships, and a growing number of human-intensive processes from existing SSIs; top-down governance doesn’t fit culturally or technologically with new engineering processes.
- The demands and pressures from modern software delivery practices, be they cultural such as Agile and DevOps, or technology-based such as cloud- and orchestration-based; gates and checkpoints built for maximum assurance often cause unacceptable disruption in processes built for speed.

- The shift to engineer self-service, typically seen as self-service IT (cloud), configuration and deployment (DevOps), and development (open source use and continuous integration); the ability to instantiate infrastructure and pipelines is also the ability to integrate your own security tools and configurations.

This new software security effort is frequently happening independently from the lessons learned that an experienced SSG might provide. In addition, this effort is driving many application lifecycle processes ever faster, regardless of whether the organization is ready to do software security risk management at that speed.

The governance-oriented approach we’ve seen for years, along with this new wave of engineering-oriented efforts, are increasingly coexisting within the same organization. In addition, they often have competing objectives, which is pulling traditional governance-driven programs into modern and evolving hybrids. Figure 11 shows this ongoing SSG evolution.

The important lesson here is that this is likely happening in your organization as well—perhaps narrowly in a few development teams or perhaps broadly as a culture shift across all of engineering. Taking an SSI to the maturing stage—and possibly to enabling, as well—requires acknowledging this engineering effort and building bridges between all stakeholders who have ownership of the different aspects of software security. It also requires acknowledging that these different stakeholders have different business objectives and different views of risk, risk management, and risk tolerance relative to those objectives. Ensuring that everyone can meet their objective while also keeping the organization safe is a major goal for every SSI.

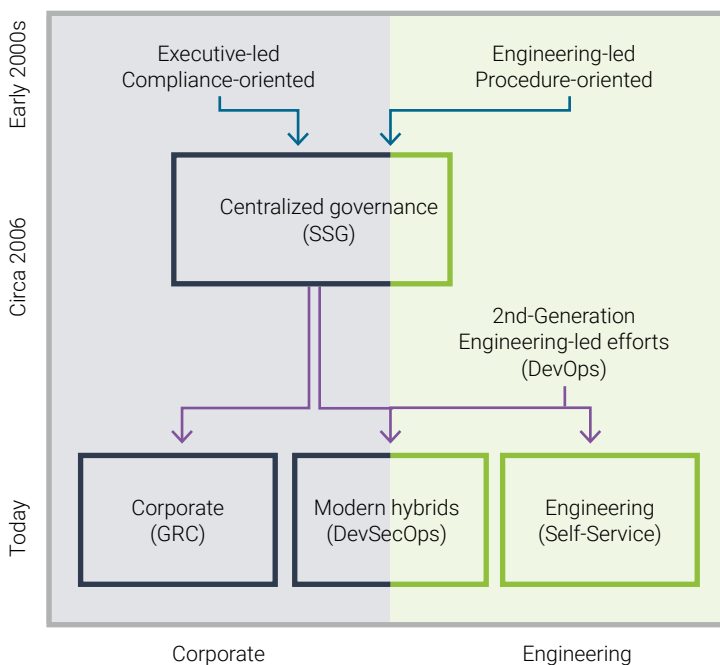


FIGURE 11. SSG EVOLUTION. These groups might have started in corporate or in engineering but, in general, settled on enforcing compliance with tools. The new wave of engineering efforts is shifting where SSGs live, what they focus on, who is accountable for what, and how stakeholders work together.

Understanding More About DevOps

The DevOps movement has highlighted the tensions between established SSIs and engineering efforts that are addressing software security their way in their own processes. Given different objectives, we find that the outcomes desired by these two approaches are usually very different. Rather than the top-down, compliance-driven style of governance-minded teams, these newer engineering-minded teams are more likely to prototype good ideas for securing software, which results in the creation of even more code and infrastructure on the critical path to delivery (e.g., security features, home-spun vulnerability discovery, security guardrails). Here, security is just another aspect of quality, and availability is just another aspect of resilience.

To keep pace with both software development process changes (e.g., CI/CD adoption) and technology architecture changes (e.g., cloud, container, and orchestration adoption), engineering efforts are independently evolving both how they apply software security activities and, in some cases, what activities they apply. The changes these engineering teams are making include downloading and integrating their own security tools, spinning up self-service cloud infrastructure and virtual assets as they need them, following policy on the use of open source software in applications but routinely downloading many other open source packages to build and manage software and processes, and so on. Engineering efforts and their associated fast-paced evolutionary changes are putting governance-driven SSIs in a race to retroactively document, communicate, and even automate the knowledge they hold so it can be useful to everyone.

Cloud service providers, software pipeline and orchestration platforms, and even QA tools have also begun adding their view of software security in their feature sets. For example, organizations are seeing platforms like GitHub, Azure DevOps, and GitLab beginning to compete by using security as a differentiator. Evolving vendor-provided features might be signaling to both the marketplace and adopting organizations that vendors believe security must be included in developer tools and that engineering security initiatives should feel comfortable relying on these external platforms as the basis of their security telemetry and even their governance workflows.

Again, the important lesson is that this is likely happening in your organization as well. Your path to an emerging or mature SSI must account for this federation of software security responsibilities and use of external providers, and enable every stakeholder to meet their business and security objectives.

Convergence as a Goal

We frequently observe governance-oriented SSIs planning centrally, seeking to proactively define an ideal risk posture during their emerging or early maturity phases. Initial uptake of the provided controls (e.g., security testing) is usually by those teams that have experienced real security issues and are looking for help, while other teams might take a wait-and-see approach.

We also observe that engineering efforts prototype controls incrementally, building on existing tools and techniques that already drive software delivery. Gains happen quickly in these emerging efforts, perhaps given the steady influx of new tools and techniques introduced by engineering but also helped along by the fact that each team is usually working in a homogenous culture on a single application and technology stack. Even so, these groups sometimes struggle to institutionalize durable gains, usually because the engineers have not yet been able to turn capability into either secure-by-default functionality or automation-friendly assurance—at least not beyond the most frequently encountered security issues and beyond their own spheres of influence.

Engineering groups tend to view security as an enabler of software features and code quality. These groups recognize the need for having security standards but tend to prefer incremental steps toward governance-as-code as opposed to a large-manual-steps-with-human-review approach to enforcement. This tends to result in engineers building security features and frameworks into architectures, automating defect discovery techniques within a software delivery pipeline, and treating security defects like any other defect. Traditional human-driven security decisions are modeled into a software-defined workflow as opposed to being written into a document and implemented in a separate risk workflow handled outside of engineering. In this type of culture, it's not that the traditional SDLC gates and risk decisions go away, it's that they get implemented differently and usually have different goals compared to those of the governance groups. SSGs, and likely champions groups as well, that

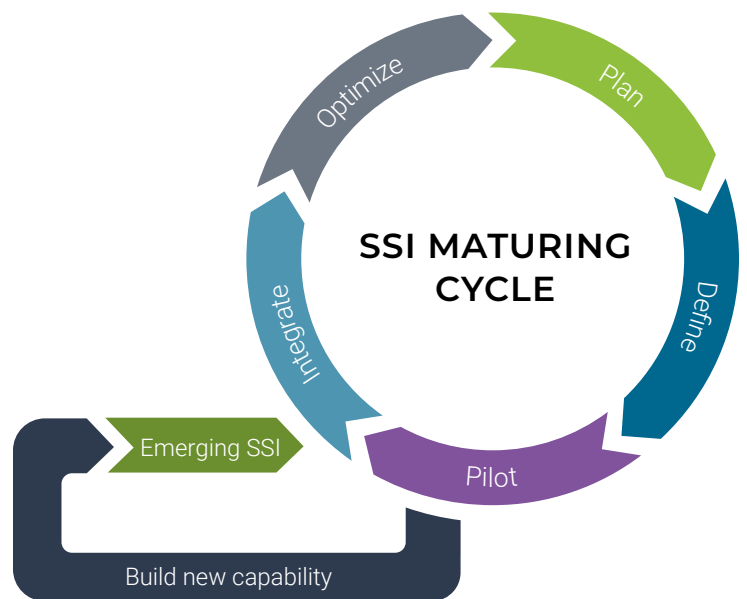


FIGURE 12. MOVING FROM EMERGING TO MATURING. Building an emerging SSI usually focuses on collecting activities into a single program. Moving from emerging to maturing requires ongoing iterative improvements and expansions. Piloting new capabilities (e.g., security champions, software supply chain risk management) likely requires reapplying the emerging approach for a specific set of activities.

begin to support this approach will speed up both convergence of various efforts and alignment with corporate risk management goals.

To summarize the lessons from the community, scaling an emerging SSI across a software portfolio is hard for everyone, and stakeholders need to understand the lessons above before investing heavily in the journey from emerging to maturing. Today's evolving cultural and technological environments require a concerted effort at converging governance and engineering objectives to create a cohesive SSI that ensures the software portfolio is appropriately protected.

MATURING AN SSI: HARMONIZING OBJECTIVES

This section provides an expanded view of an SSI journey. With the foundations established, SSG leaders shift their attention to scaling risk-based controls across the entire software portfolio and enabling development to find and fix issues early in the software lifecycle. The SSI has likely reached the emerging stage across multiple capabilities (see Figure 10) and is maturing specific aspects of its initiative. That maturing includes both adding new activities and scaling existing ones (see Figure 12). It especially includes building bridges between various software security efforts in corporate and engineering groups.

This section on maturing an SSI repeats some of the foundational BSIMM activities from the "Starting an SSI: Getting to an Emerging State" section. We do this because most organizations won't treat SSI creation as a waterfall process. Instead, they will, for example, establish policy, set up a champions program, deploy defect discovery tools, and so on in overlapping, incremental improvement cycles. In addition, many organizations will determine in the emerging phase that some activities can wait a bit while engaging in other, more necessary, software security efforts. In either case, this is a good place for a reminder to keep working on foundational activities.

Establish Leadership and Objectives

Ensure that there is a single SSI and provide the proper resources for the owner tasked with shepherding the organization so the group can meet risk management objectives. At this point, the SSI might include multiple SSGs and owners (e.g., across major products or business units), and working to harmonize these efforts must be a key goal. Ensure that the SSI is supported by a full-time team—an SSG—that can scale across the organization. Establishing this structure might not involve hiring staff immediately, but it will likely entail assembling a full-time team to implement key foundational activities central to supporting the assurance objectives further defined and institutionalized in policy [CP1.3], standards [SR1.1], and processes [SM1.1].

The SSG will require a mix of skills, including technical security knowledge, scripting and coding experience, and architectural skill. As organizations migrate toward their view of DevSecOps, the SSG might build its own software in the form of security automation, defect discovery in CI/CD pipelines, and infrastructure- and governance-as-code. SSGs often need to mentor, train, and work directly with developers, so communication skills, teaching ability, and practical knowledge are must-haves for at least some SSG

staff. Essentially, the SSG is a group of people—whether one person, 10, or 100—who must improve the security posture of the software portfolio and all the processes that generate it, so management skills, risk management perspectives, an ability to contribute to engineering value streams, and an ability to break silos are critical success factors.

Within engineering teams, we see individuals taking on leadership roles such as product security engineer or security architect, while possessing functional titles such as Site Reliability Engineer, DevOps Engineer, or similar. Their responsibilities often include comparison and selection of security tools, definition of secure design guidelines and acceptable remediation actions, and implementation of infrastructure-as-code for secure packaging, delivery, and operations. Harmonizing leadership views across the SSG and engineering is a critical step to success.

Expand Security Controls

Use existing knowledge to choose the important software security activities to initiate, scale, or improve. This knowledge includes SSI scope, compliance, technology stacks, and deployment models, as well as the issues uncovered in defect discovery efforts. Common activity choices are policy [CP1.3], SDLC checkpoint conditions [SM1.4], testing [AA1.2, CR1.4, ST1.4, PT1.3, SR2.4], and training [T1.7], which are typically built out in a quick-win approach. When choosing and implementing new controls, it's often easier to get buy-in by showing adherence to well-known guidance (e.g., BSIMM, NIST SSDF, regulators) or choosing security controls that align with general industry guidance (e.g., OWASP, CWE, analysts). Ensure that activity selection includes an appropriate mix of preventive [SR1.1, SFD2.1] and detective controls (e.g., testing) to maximize positive impact on the organization's risk posture.

Essentially, the SSG is a group of people—whether one person, 10, or 100—who must improve the security posture of the software portfolio.

Engage Development

As noted throughout this section, engineering teams are likely already thinking about various aspects of security related to design, configuration, infrastructure, and deployment. Engaging development begins by creating mutual awareness of how the SSG and development teams see the next steps in maturing security efforts. Successfully engaging early on relies on bridge-building and credentialing the SSG as competent in development culture, toolchains, and technologies. It also includes building awareness around which security capabilities constitute an SSDL and beginning to determine how those capabilities are expected to be conducted. Building consensus on what role each department will play in improving capabilities over the next evolutionary cycle greatly facilitates success.

To facilitate tool adoption, the SSG might dedicate some portion of their efforts or build a team of security champions [SM2.3] to serve as tool mentors to help development teams not only integrate the tools but also triage and interpret results [CR1.7]. Although the primary objective is to embed security leadership inside development, these individuals also serve as both key points of contact and interface points for the SSG to interact with engineering teams and monitor progress. Because they are local to teams, champions also facilitate defect management goals, such as tracking recurring issues to drive remediation [PT1.2]. The SSG can also roll out software security training [T2.9] tailored to the most common security defects identified through AST, often cataloged by technology stack and coding language.

Inventory and Select In-Scope Software

Take an enterprise-wide perspective when building a view into the software portfolio. Engaging directly with application business owners by, for example, using questionnaire-style data gathering is a good start. It's useful to focus on applications (with owners who are responsible for risk management) as the initial unit of inventory measure, but remember that many vital software components aren't applications (e.g., libraries, APIs, scripts, pipeline tests, infrastructure-as-code). In addition to understanding application characteristics (e.g., programming language, architecture type such as web or mobile, the revenue generated) as a view into risk, capture and maintain the same information for all software. Focus on understanding where sensitive data resides and flows (e.g., PII inventory) [CP2.1] along with the status of active development projects.

Rather than taking an organizational structure and owner-based view, engineering teams usually attempt to understand software inventory by extracting it from the same tools they use to manage their IT assets. They usually combine two or more of the following approaches to software inventory creation:

- Discovery, import, and visualization of assets managed by the organization's cloud and data center virtualization management consoles
- Scraping and extracting assets and tags from infrastructure-as-code held in code repositories, as well as processing metadata from container and other artifact registries
- Outside-in web and network scanning for publicly discoverable assets, connectivity to known organizational assets, and related ownership and administrative information

With a software inventory in hand, impose security requirements using formalized risk-based approaches to cover as much of the software portfolio as possible. Using simple criteria (e.g., software size, regulatory constraints, internal vs. external facing, data classification), assign a risk classification (e.g., high, medium, low) to each application [AA1.4]. Define the initial set of software and project teams with which to prototype security activities. Although application risk classifications are often the primary driver, we have observed firms using other information, such as whether a major change in application architecture is being undertaken (e.g., shift to a cloud-native architecture) or whether the software contains critical code (e.g., cryptography, proprietary business logic). Firms find it

CHECKLIST FOR MATURING SSIs

- 1. Establish Leadership and Objectives.** Formalize staffing, objectives, budgets, and approach, then tell everybody about it.
- 2. Expand Security Controls.** Increase program impact through policy, testing, training, and other quick wins.
- 3. Engage Development.** Use security champions to build bridges and harmonize software security objectives.
- 4. Inventory and Select In-Scope Software.** Expand the application inventory to include all software, not just applications.
- 5. Enforce Security Basics Everywhere.** Use automation to ensure that you run software only on good systems (cloud or otherwise).
- 6. Integrate Defect Discovery and Prevention.** Use automation and integration to scale and shift defect discovery and prevention everywhere.
- 7. Upgrade Incident Response.** Ensure that software security experts are involved in all software security events and improve the program from lessons learned.
- 8. Repeat and Improve.** Growth does not happen in a straight line. You will have to revisit, remeasure, and re-plan multiple times.

beneficial to include in the selection process some engineering teams that are already doing some security activity organically.

Engineering teams might have a different idea of what in-scope software means relative to the security efforts they already have underway—if they're working on one application, then that application is likely to be their scope. When required to prioritize specific applications' components, we observe engineering teams using the following as input:

- **Velocity.** Teams conducting active new development or major refactoring
- **Regulation.** Those services or data repositories to which specific development or configuration requirements for security or privacy apply [CP1.1, CP1.2]
- **Opportunity.** Software that solves critical technical challenges or that adopts key technologies

Prioritized software is then usually the target for test automation [ST2.5], vulnerability discovery tooling, or security features [SFD1.1].

Enforce Security Basics Everywhere

Commonly observed today regardless of SSG age are basic security controls enforced in hosts and networks [SE1.2] and in cloud environments [SE1.3]. A common strength for organizations that have good controls over the infrastructure assets they manage, these basics are accomplished through a combination of IT

provisioning controls, written policy, prebuilt and tested golden images, sensors and monitoring capabilities, server hardening and configuration standards, infrastructure-as-code, and entire groups dedicated to patching. As firms migrate private infrastructure to cloud environments, organizations must carefully reestablish their assurance-based controls to maintain and verify adherence to security policy. To keep tabs on the growing number of virtual assets created by engineering groups and their automation, organizations often must deploy custom solutions [AM3.3] to overcome limitations in a cloud provider's ability to meet desired policy.

Organizations rarely move their entire SSI from emerging to enabling all at once.

Governance and engineering teams often cooperate to build in enforced security basics for infrastructure and cloud environments, leveraging containers [SE2.5], infrastructure-as-code [SE2.2], and orchestration [SE2.7]. Over time, these security basics expand to include internal development environments, toolchains, deployment automation, code repositories, and other important infrastructure.

Integrate Defect Discovery and Prevention

Initial defect discovery efforts tend to be one-off (by using centralized commercial tools [CR1.2]) and to target the most critical applications, with a plan to scale efforts over time. Scaling prioritization might be selected for compliance or contractual reasons, or because it applies to a phase of the software lifecycle (e.g., shift everywhere to do threat modeling at design time [AA1.1], composition analysis on software repositories [SE3.8], SAST during development [CR1.4], DAST in preproduction [ST1.4], and penetration testing on deployed software [PT1.1, PT1.3]). The point is to automate and scale the chosen defect discovery activities. However, scaling through automation and integration must come without disrupting CI/CD pipelines (e.g., due to tools having long execution times), without generating large volumes of perceived false positives, and without impeding delivery velocity (e.g., through opaquely breaking builds or denying software promotion) except under clear or agreed-upon circumstances.

In addition to defect discovery, engineering teams might favor prevention controls they can apply to software directly in the form of security features [SFD1.1]. These controls can take the form of microservices (e.g., authentication or other identity and access management) [SE2.5], common product libraries (e.g., encryption) [SFD2.1], or even infrastructure security controls (e.g., controlling scope of access to production secrets through vault technologies).

Some engineering groups have taken steps to tackle the prevention of certain classes of vulnerability in a wholesale manner [CMVM3.1],

using development frameworks that preclude them. Ask security-minded engineers for their opinion about framework choices and empower them to incorporate their understanding of security features and security posture tradeoffs.

Upgrade Incident Response

Ensure that defined incident response processes include SSG representation [CMVM1.1]. Determining whether an incident has software security roots requires specific skills that are not often found in traditional IT groups. Work with engineering teams, especially DevOps engineers, to help make the connections between those events and alerts raised in production and the associated artifacts, pipelines, repositories, and responsible teams. This traceability allows these groups to effectively prioritize security issues on which the SSG will focus. Feedback from the field on what is happening greatly enhances the top N lists ([AM2.5, CR2.7]) that many organizations use to help establish priorities.

Security engineers who are in development teams and more familiar with application logic might be able to facilitate instructive monitoring and logging. They can coordinate with DevOps engineers to generate in-application defenses that are tailored for business logic and expected behavior, therefore likely being more effective than, for example, WAF rules. Introducing such functionality will in turn provide richer feedback and allow a more tailored response to application behavior [SE3.3].

Organizations deploying cloud-native applications using orchestration might respond to incidents (or to data indicating imminent incidents) with an increase in logging, perhaps by adjusting traffic to the distribution of image types in production. Much of this is possible only with embedded security engineers who are steeped in the business context of a development team and have good relationships with that team's DevOps engineers; satellite members (security champions) can be a good resource for these individuals. Under these circumstances, incident response moves at the speed of a well-practiced single team [CMVM2.1] rather than that of an interdepartmental playbook.

Repeat and Improve

As noted earlier, working through activity growth for emerging and maturing SSIs probably won't happen in a straight line. There'll be changes in priorities, resources, and responsibilities, along with changes in attackers, attacks, technologies, and everything else. It's necessary to take time periodically to determine how well the SSI is performing against business objectives and adjust as necessary.

As a reminder, organizations rarely move their entire SSI from emerging to enabling all at once. Different parts of the SSI will shift between emerging, maturing, and enabling a few times over the years with different timing that SSG leaders will need to plan for.

ENABLING SSIs

Achieving software security scale—of expertise, portfolio coverage, tool integration, vulnerability discovery accuracy, process consistency, and so on—remains a top priority. However, firms often scale one or two capabilities (e.g., defect discovery, training) but fail to scale others (e.g., architecture analysis, vendor management). Given mature activities, there's a treasure trove of data to be harvested and included in KPI and KRI reporting dashboards. Then executives start asking the very difficult questions: Are we getting better? Is our implementation working well? Where are we lagging? How can we go faster with less overhead? What's our message to the Board? The efficacy of an SSI will be supported by ongoing data collection and metrics reporting that seeks to answer such questions [SM3.3].

Progress Isn't a Straight Line

As mentioned earlier, organizations don't always progress from maturing to enabling in one try or on a straight path, for example, some SSI capabilities might be enabling while others are still emerging or maturing. Based on our experience, firms with some portion of their SSI operating in an enabling state have likely been in existence for longer than three years. Although we don't have enough data to generalize enabling SSIs, we do see common themes for those that strive to reach this state:

- **Top N Risk Reduction.** Everyone relentlessly identifies and closes top N weaknesses, placing emphasis on obtaining visibility into all sources of vulnerability, whether in-house developed code, open source code [SR2.7], vendor code [SR3.2], toolchains, or any associated environments and processes [SE1.2, SE1.3]. These top N weaknesses are most useful when specific to the organization, evaluated at least annually, and tied to metrics to prioritize SSI efforts that improve risk posture.
- **Tool Customization.** Security leaders place a concerted effort into tuning tools (e.g., customization for static analysis, fuzzing, penetration testing) to improve integration, accuracy, consistency, and depth of analysis [CR2.6, ST2.6, AM3.2, PT3.2]. Customization focuses not only on improving result fidelity and applicability across the portfolio but also on pipeline integration and timely execution, improving ease of use for everyone.
- **Feedback Loops.** Loops are created between SSDL activities to improve effectiveness as deliverables from activities ebb and flow with each other. For example, an expert within QA might leverage architecture analysis results when creating security test cases [ST3.3]. Likewise, feedback from the field might be used to drive SSDL improvement through enhancements to a hardening standard [CMVM3.2]. The concept of routinely conducting blameless postmortems to find root causes and drive remediation seems to be gaining ground in some firms.
- **Data-Driven Governance.** The more mature groups instrument everything to collect data that in turn becomes metrics for measuring SSI efficiency and effectiveness against KRIs and KPIs [SM3.3]. As an example, a metric such as defect density might be leveraged to track performance of individual business units and application teams. Metrics choices are very specific to each organization and also evolve over time.

Push for Agile-Friendly SSIs

In recent years, we've observed governance-oriented teams—often out of necessity to remain in sync with engineering teams—evolving to become more Agile-friendly:

- Putting “Sec” in DevOps is becoming a mission-critical objective. SSG leadership routinely partners with IT, cloud, development, QA, and operations leadership to ensure that the SSI mission aligns with DevOps values and principles.
- SSG leaders realize they need in-house talent with coding expertise to improve not only their credibility with engineering but also their understanding of modern software delivery practices. Job descriptions for SSG roles now mention experience and qualification requirements such as cloud, mobile, containers, and orchestration security, as well as coding. We expect this list to grow as other topics become more mainstream, such as architecture and testing requirements around serverless computing and single-page application approaches.
- To align better with DevOps values (e.g., agility, collaboration, responsiveness), SSG leaders are beginning to replace traditional people-driven activities with people-optional, pipeline-driven automated tasks. This often comes in the form of automated security tool execution, bugs filed automatically to defect notification channels, builds flagged for critical issues, and automated triggers to respond to real-time operational events.
- Scaling outreach and expertise through the implementation of an ever-growing satellite is viewed as a short-term rather than long-term goal. Organizations report improved responsiveness and engagement as part of DevOps initiatives when they've localized security expertise in the engineering teams. Champions are also becoming increasingly sophisticated in building reusable artifacts (e.g., security sensors) in development and deployment streams to directly support SSI activities.
- SSG leaders are partnering with operations to implement application-layer production monitoring and automated mechanisms for responding to security events. There is a high degree of interest in consuming real-time security events for data collection and analysis to produce useful metrics.

In summary, engineering teams have likely taken an enabling approach from the beginning. Their security efforts are contributions from engineers who deliver software early and often, constantly improving it rather than relying on explicit strategy backed by top-down policies. They make their software available to everyone to prevent future issues and use evangelism to encourage uptake. They review production failures and make changes, often with automation, to their toolchains and processes. That said, perceptions of business and technical risk between corporate and engineering groups often differ in substantial ways. Bringing the groups together to share responsibilities for software security, as well as definitions of and goals for needed risk management, while enabling broad stakeholder productivity is a primary goal for any SSI.

C. DETAILED VIEW OF THE BSIMM FRAMEWORK

- The BSIMM framework and data model evolve over time to accurately represent actual software security practices. Understanding these changes will help you set strategic directions for your own SSI.

In Part 5, we introduced the BSIMM framework. Here, we explore it in more detail, including the methodology of how we created the model, how it evolved over time, and how we updated it for BSIMM13.

As a descriptive model, the only goal of the BSIMM is to observe and report. We like to say we visited many restaurants to see what was happening and observed that “there are three chicken eggs in an omelet.” Note that the BSIMM does not extrapolate to say, “all omelets must have three eggs,” “only chicken eggs make acceptable omelets,” “omelets must be eaten every day,” or any other value judgements. We offer simple observations, simply reported.

Of course, during our assessment efforts across hundreds of organizations, we also make qualitative observations about how SSIs are evolving and report many of those as trends, insights, analysis, and other topical discussions both in this document and within the BSIMM community.

Our “just the facts” approach is hardly novel in science and engineering, but in the realm of software security, it has not previously been applied at this scale. Other work around SSI modeling has either described the experience of a single organization or offered prescriptive guidance based on a combination of personal experience and opinion.

During our assessment efforts across hundreds of organizations, we make qualitative observations about how SSIs are evolving and report many of those as insights, analysis, and other discussions in this document and in the BSIMM online community.

THE BSIMM SKELETON

The BSIMM skeleton provides a way to view the model at a glance and is useful when assessing an SSI. The skeleton is shown in Figure 13, organized by domains and practices. More complete descriptions of the activities and examples are available in Part 6 of this document.

CREATING BSIMM13 FROM BSIMM12

BSIMM13 includes updated activity descriptions, data from firms in multiple vertical markets, and a longitudinal study. For BSIMM13, we added 27 firms and removed 25, resulting in a data pool of 130 firms. In addition, in the time since we launched BSIMM12, 17 firms conducted reassessments to update their scorecards, and we assessed additional business units for five firms.

As shown below, we used the resulting observation counts to refine activity placement in the framework, which resulted in moving six activities to different levels. In addition, we added three newly observed activities, resulting in a total of 125 activities in BSIMM13:

- [T3.3 Host software security events] became T2.10
- [T3.4 Require an annual refresher] became T2.11
- [SR3.1 Control open source risk] became SR2.7
- [AA1.3 Have SSG lead design review efforts] became AA2.4
- [CR1.6 Use centralized defect reporting to close the knowledge loop] became CR2.8
- [SE2.6 Implement cloud security controls] became SE1.3
- [SM3.5 Integrate software supply chain risk management] was added to the model
- [SE3.8 Perform application composition analysis on code repositories] was added to the model
- [CMVM3.8 Do attack surface management for deployed applications] was added to the model

We also carefully considered but did not adjust [CP2.5 Ensure executive awareness of compliance and privacy obligations] or [AM3.3 Monitor automated asset creation] at this time; we will do so if the observation rates continue to increase.

GOVERNANCE					
STRATEGY & METRICS		COMPLIANCE & POLICY		TRAINING	
[SM1.1]	Publish process and evolve as necessary.	[CP1.1]	Unify regulatory pressures.	[T1.1]	Conduct software security awareness training.
[SM1.3]	Educate executives on software security.	[CP1.2]	Identify privacy obligations.	[T1.7]	Deliver on-demand individual training.
[SM1.4]	Implement security checkpoints and associated governance.	[CP1.3]	Create policy.	[T1.8]	Include security resources in onboarding.
[SM2.1]	Publish data about software security internally and use it to drive change.	[CP2.1]	Build a PII inventory.	[T2.5]	Enhance satellite (security champions) through training and events.
[SM2.2]	Enforce security checkpoints and track exceptions.	[CP2.2]	Require security sign-off for compliance-related risk.	[T2.8]	Create and use material specific to company history.
[SM2.3]	Create or grow a satellite (security champions).	[CP2.3]	Implement and track controls for compliance.	[T2.9]	Deliver role-specific advanced curriculum.
[SM2.6]	Require security sign-off prior to software release.	[CP2.4]	Include software security SLAs in all vendor contracts.	[T2.10]	Host software security events.
[SM2.7]	Create evangelism role and perform internal marketing.	[CP2.5]	Ensure executive awareness of compliance and privacy obligations.	[T2.11]	Require an annual refresher.
[SM3.1]	Use a software asset tracking application with portfolio view.	[CP3.1]	Document a software compliance story.	[T3.1]	Reward progression through curriculum.
[SM3.2]	Make SSI efforts part of external marketing.	[CP3.2]	Ensure compatible vendor policies.	[T3.2]	Provide training for vendors and outsourced workers.
[SM3.3]	Identify metrics and use them to drive resourcing.	[CP3.3]	Drive feedback from software lifecycle data back to policy.	[T3.5]	Provide expertise via open collaboration channels.
[SM3.4]	Integrate software-defined lifecycle governance.			[T3.6]	Identify new satellite members (security champions) through observation.
[SM3.5]	Integrate software supply chain risk management.				

INTELLIGENCE					
ATTACK MODELS		SECURITY FEATURES & DESIGN		STANDARDS & REQUIREMENTS	
[AM1.2]	Use a data classification scheme for software inventory.	[SFD1.1]	Integrate and deliver security features.	[SR1.1]	Create security standards.
[AM1.3]	Identify potential attackers.	[SFD1.2]	Application architecture teams engage with the SSG.	[SR1.2]	Create a security portal.
[AM1.5]	Gather and use attack intelligence.	[SFD2.1]	Leverage secure-by-design components and services.	[SR1.3]	Translate compliance constraints to requirements.
[AM2.1]	Build attack patterns and abuse cases tied to potential attackers.	[SFD2.2]	Create capability to solve difficult design problems.	[SR2.2]	Create a standards review process.
[AM2.2]	Create technology-specific attack patterns.	[SFD3.1]	Form a review board or central committee to approve and maintain secure design patterns.	[SR2.4]	Identify open source.
[AM2.5]	Maintain and use a top N possible attacks list.	[SFD3.2]	Require use of approved security features and frameworks.	[SR2.5]	Create SLA boilerplate.
[AM2.6]	Collect and publish attack stories.	[SFD3.3]	Find and publish secure design patterns from the organization.	[SR2.7]	Control open source risk.
[AM2.7]	Build an internal forum to discuss attacks.			[SR3.2]	Communicate standards to vendors.
[AM3.1]	Have a research group that develops new attack methods.			[SR3.3]	Use secure coding standards.
[AM3.2]	Create and use automation to mimic attackers.			[SR3.4]	Create standards for technology stacks.
[AM3.3]	Monitor automated asset creation.				

SSDL TOUCHPOINTS					
ARCHITECTURE ANALYSIS		CODE REVIEW		SECURITY TESTING	
[AA1.1]	Perform security feature review.	[CR1.2]	Perform opportunistic code review.	[ST1.1]	Perform edge/boundary value condition testing during QA.
[AA1.2]	Perform design review for high-risk applications.	[CR1.4]	Use automated code review tools.	[ST1.3]	Drive tests with security requirements and security features.
[AA1.4]	Use a risk methodology to rank applications.	[CR1.5]	Make code review mandatory for all projects.	[ST1.4]	Integrate opaque-box security tools into the QA process.
[AA2.1]	Perform architecture analysis using a defined process.	[CR1.7]	Assign code review tool mentors.	[ST2.4]	Drive QA tests with AST results.
[AA2.2]	Standardize architectural descriptions.	[CR2.6]	Use custom rules with automated code review tools.	[ST2.5]	Include security tests in QA automation.
[AA2.4]	Have SSG lead design review efforts.	[CR2.7]	Use a top N bugs list (real data preferred).	[ST2.6]	Perform fuzz testing customized to application APIs.
[AA3.1]	Have engineering teams lead AA process.	[CR2.8]	Use centralized defect reporting to close the knowledge loop.	[ST3.3]	Drive tests with design review results.
[AA3.2]	Drive analysis results into standard architecture patterns.	[CR3.2]	Build a capability to combine AST results.	[ST3.4]	Leverage code coverage analysis.
[AA3.3]	Make the SSG available as an AA resource or mentor.	[CR3.3]	Create capability to eradicate bugs.	[ST3.5]	Begin to build and apply adversarial security tests (abuse cases).
		[CR3.4]	Automate malicious code detection.	[ST3.6]	Implement event-driven security testing in automation.
		[CR3.5]	Enforce secure coding standards.		

DEPLOYMENT					
PENETRATION TESTING		SOFTWARE ENVIRONMENT		CONFIGURATION MANAGEMENT & VULNERABILITY MANAGEMENT	
[PT1.1]	Use external penetration testers to find problems.	[SE1.1]	Use application input monitoring.	[CMVM1.1]	Create or interface with incident response.
[PT1.2]	Feed results to the defect management and mitigation system.	[SE1.2]	Ensure host and network security basics are in place.	[CMVM1.2]	Identify software defects found in operations monitoring and feed them back to development.
[PT1.3]	Use penetration testing tools internally.	[SE1.3]	Implement cloud security controls.	[CMVM2.1]	Have emergency response.
[PT2.2]	Penetration testers use all available information.	[SE2.2]	Define secure deployment parameters and configurations.	[CMVM2.2]	Track software bugs found in operations through the fix process.
[PT2.3]	Schedule periodic penetration tests for application coverage.	[SE2.4]	Protect code integrity.	[CMVM2.3]	Develop an operations software inventory.
[PT3.1]	Use external penetration testers to perform deep-dive analysis.	[SE2.5]	Use application containers to support security goals.	[CMVM3.1]	Fix all occurrences of software bugs found in operations.
[PT3.2]	Customize penetration testing tools.	[SE2.7]	Use orchestration for containers and virtualized environments.	[CMVM3.2]	Enhance the SSDL to prevent software bugs found in operations.
		[SE3.2]	Use code protection.	[CMVM3.3]	Simulate software crises.
		[SE3.3]	Use application behavior monitoring and diagnostics.	[CMVM3.4]	Operate a bug bounty program.
		[SE3.6]	Create bills of materials for deployed software.	[CMVM3.5]	Automate verification of operational infrastructure security.
		[SE3.8]	Perform application composition analysis on code repositories.	[CMVM3.6]	Publish risk data for deployable artifacts.
				[CMVM3.7]	Streamline incoming responsible vulnerability disclosure.
				[CMVM3.8]	Do attack surface management for deployed applications.

FIGURE 13. THE BSIMM SKELETON. Within the SSF, the 125 activities are organized into 12 practices within the four BSIMM domains.

As concrete examples of how the BSIMM functions as an observational model, consider the activities that are now SM3.3 and SR3.3, which both started as level 1 activities. The BSIMM1 activity [SM1.5 Identify metrics and use them to drive resourcing] became SM2.5 in BSIMM3 and is now SM3.3 due to its observation rate remaining fairly static while other activities in the practice became observed much more frequently. Similarly, the BSIMM1 activity [SR1.4 Use secure coding standards] became SR2.6 in BSIMM6 and is now SR3.3 as its observation rate has decreased.

In BSIMM13, we have the first activity that migrated from level 3 to level 1—[SE1.3 Implement cloud security controls], which was introduced in BSIMM9. While the relative growth of [SE2.5 Use application containers to support security goals] has slowed down, it is one of the potential candidates to migrate from level 3 to level 1 over the next couple of years. See Table 5 for the observation growth in activities that were added since BSIMM7.

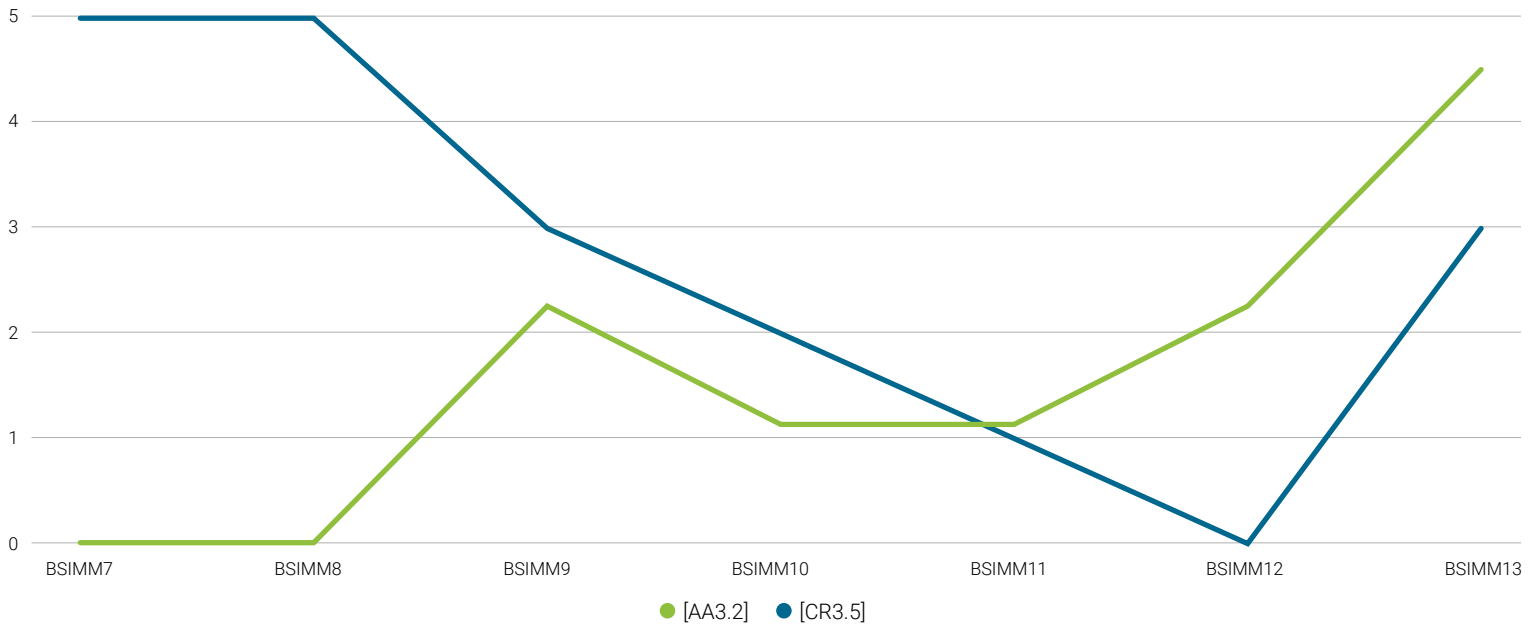


FIGURE 14. NUMBER OF OBSERVATIONS FOR [AA3.2] AND [CR3.5] OVER TIME. [AA3.2 Drive analysis results into standard design patterns] had zero observations during BSIMM7 and BSIMM8, while [CR3.5 Enforce secure coding standards] decreased to zero observations from BSIMM8 to BSIMM12 (the number of observations increased back to three in BSIMM13). Currently, the only activities with zero observations are the three activities added in BSIMM13.

OBSERVATIONS							
ACTIVITY	BSIMM7	BSIMM8	BSIMM9	BSIMM10	BSIMM11	BSIMM12	BSIMM13
SE3.4 (now SE2.5)	0	4	11	14	31	44	52
SE3.5 (now SE2.7)			0	5	22	33	42
SE3.6			0	3	12	14	18
SE3.7 (now SE1.3)			0	9	36	59	79
SM3.4				0	1	6	5
AM3.3				0	4	6	11
CMVM3.5				0	8	10	13
ST3.6					0	2	3
CMVM3.6					0	0	3
CMVM3.7						0	20
SM3.5							0
SE3.8							0
CMVM3.8							0

TABLE 5. NEW ACTIVITIES. Some activities have seen exceptional growth (highlighted in orange) in observation counts, likely demonstrating their widespread utility. [SE3.7], highlighted in gray, is the first activity to migrate from level 3 (very uncommon) to level 1 (common).

WHERE DO OLD ACTIVITIES GO?

We continue to ponder the question, “Where do activities go when no one does them anymore?” In addition to [CR3.5 Enforce secure coding standards] (shown in Figure 14), we’ve noticed that the observation rate for other seemingly useful activities has decreased significantly in recent years:

- [T3.6 Identify new satellite members (security champions) through observation] observed in 11 of 51 firms in BSIMM4 but only in seven of 130 firms in BSIMM13
- [SFD3.3 Find and publish secure design patterns from the organization] observed in 14 of 51 firms in BSIMM4 but only in seven of 130 firms in BSIMM13
- [SR3.3 Use secure coding standards] observed in 23 of 78 firms in BSIMM6 but only in 17 of 130 firms in BSIMM13

We believe there are two primary reasons why observations for some activities have decreased toward zero over time. First, some activities have become part of the culture and drive different behavior—for example, choosing satellite members might become a more organic part of the SSDL without requiring extra effort in identifying satellite members [T3.6 Identify new satellite members (security champions) through observation] to grow that team [SM2.3 Create or grow a satellite (security champions)]. Second, some activities don’t yet fit tightly with the evolving

engineering culture, and the activity effort currently causes too much friction. For example, continuously going to engineering teams to find secure design patterns [SFD3.3 Find and publish secure design patterns from the organization] might unacceptably delay key development processes.

It might also be the case that evolving SSI and DevOps architectures are changing the way some activities are getting done. If an organization’s use of purpose-built architectures, development kits, and libraries is sufficiently consistent, for example, perhaps it’s less necessary to lean on prescriptive coding standards [CR3.5 Enforce secure coding standards] as a measure of acceptable code.

As a point of culture-driven contrast, we see significant increases in observation counts for activities such as [SE1.3 Implement cloud security controls], [SE2.5 Use application containers to support security goals], and [SE2.7 Use orchestration for containers and virtualized environments], likely for similar reasons that we see lower counts for the other activities above. The engineering culture has shifted to be more self-service and to include increased telemetry that produces more data for everyone to use. We keep a close watch on the BSIMM data pool and will make adjustments as needed, which might include dropping an activity from the model.

MODEL CHANGES OVER TIME

Being a unique, real-world reflection of actual software security practices, the BSIMM naturally changes over time. While each release of the BSIMM captures the current dataset and provides the most current guidance, reflection upon past changes can help clarify the ebb and flow of specific activities. Table 6 shows the activity moves, adds, and deletes that have occurred since the BSIMM’s creation.

<p>CHANGES FOR BSIMM13 (125 ACTIVITIES)</p>	<ul style="list-style-type: none"> • T3.3 Host software security events became T2.10 • T3.4 Require an annual refresher became T2.11 • SR3.1 Control open source risk became SR2.7 • AA1.3 Have SSG lead design review efforts became AA2.4 • CR1.6 Use centralized defect reporting to close the knowledge loop became CR2.8 • SE2.6 Implement cloud security controls became SE1.3 • SM3.5 Integrate software supply chain risk management added to the model • SE3.8 Perform application composition analysis on code repositories added to the model • CMVM3.8 Do attack surface management for deployed applications added to the model
<p>CHANGES FOR BSIMM12 (122 ACTIVITIES)</p>	<ul style="list-style-type: none"> • SM1.2 Create evangelism role and perform internal marketing became SM2.7 • T1.5 Deliver role-specific advanced curriculum became T2.9 • ST2.1 Integrate black-box security tools into the QA process became ST1.4 • SE3.5 Use orchestration for containers and virtualized environments became SE2.7 • CMVM3.7 Streamline incoming responsible vulnerability disclosure added to the model
<p>CHANGES FOR BSIMM11 (121 ACTIVITIES)</p>	<ul style="list-style-type: none"> • T2.6 Include security resources in onboarding became T1.8 • CR2.5 Assign tool mentors became CR1.7 • SE3.4 Use application containers to support security goals became SE2.5 • SE3.7 Ensure cloud security basics became SE2.6 • ST3.6 Implement event-driven security testing in automation added to the model • CMVM3.6 Publish risk data for deployable artifacts added to the model

CHANGES FOR BSIMM10 (119 ACTIVITIES)	<ul style="list-style-type: none"> • T1.6 Create and use material specific to company history became T2.8 • SR2.3 Create standards for technology stacks moves to become SR3.4 • SM3.4 Integrate software-defined lifecycle governance added to the model • AM3.3 Monitor automated asset creation added to the model • CMVM3.5 Automate verification of operational infrastructure security added to the model
CHANGES FOR BSIMM9 (116 ACTIVITIES)	<ul style="list-style-type: none"> • SM2.5 Identify metrics and use them to drive resourcing became SM3.3 • SR2.6 Use secure coding standards became SR3.3 • SE3.5 Use orchestration for containers and virtualized environments added to the model • SE3.6 Enhance application inventory with operations bill of materials added to the model • SE3.7 Ensure cloud security basics added to the model
CHANGES FOR BSIMM8 (113 ACTIVITIES)	<ul style="list-style-type: none"> • T2.7 Identify new satellite through training became T3.6 • AA2.3 Make SSG available as AA resource or mentor became AA3.3
CHANGES FOR BSIMM7 (113 ACTIVITIES)	<ul style="list-style-type: none"> • AM1.1 Maintain and use a top N possible attacks list became AM2.5 • AM1.4 Collect and publish attack stories became AM2.6 • AM1.6 Build an internal forum to discuss attacks became AM2.7 • CR1.1 Use a top N bugs list became CR2.7 • CR2.2 Enforce coding standards became CR3.5 • SE3.4 Use application containers to support security goals added to model
CHANGES FOR BSIMM6 (112 ACTIVITIES)	<ul style="list-style-type: none"> • SM1.6 Require security sign-off prior to software release became SM2.6 • SR1.4 Use secure coding standards became SR2.6 • ST3.1 Include security tests in QA automation became ST2.5 • ST3.2 Perform fuzz testing customized to application APIs became ST2.6
CHANGES FOR BSIMM-V (112 ACTIVITIES)	<ul style="list-style-type: none"> • SFD2.3 Find and publish mature design patterns from the organization became SFD3.3 • SR2.1 Communicate standards to vendors became SR3.2 • CR3.1 Use automated tools with tailored rules became CR2.6 • ST2.3 Begin to build and apply adversarial security tests (abuse cases) became ST3.5 • CMVM3.4 Operate a bug bounty program added to model
CHANGES FOR BSIMM4 (111 ACTIVITIES)	<ul style="list-style-type: none"> • T2.1 Deliver role-specific advanced curriculum became T1.5 • T2.2 Company history in training became T1.6 • T2.4 Deliver on-demand individual training became T1.7 • T1.2 Include security resources in onboarding became T2.6 • T1.4 Identify new satellite members through training became T2.7 • T1.3 Establish SSG office hours became T3.5 • AM2.4 Build an internal forum to discuss attacks became AM1.6 • CR2.3 Make code review mandatory for all projects became CR1.5 • CR2.4 Use centralized reporting to close the knowledge loop became CR1.6 • ST1.2 Share security results with QA became ST2.4 • SE2.3 Use application behavior monitoring and diagnostics became SE3.3 • CR3.4 Automate malicious code detection added to model • CMVM3.3 Simulate software crises added to model
CHANGES FOR BSIMM3 (109 ACTIVITIES)	<ul style="list-style-type: none"> • SM1.5 Identify metrics and use them to drive resourcing became SM2.5 • SM2.4 Require security sign-off became SM1.6 • AM2.3 Gather and use attack intelligence became AM1.5 • ST2.2 Drive tests with security requirements and security features became ST1.3 • PT2.1 Use pen testing tools internally became PT1.3
CHANGES FOR BSIMM2 (109 ACTIVITIES)	<ul style="list-style-type: none"> • T2.3 Require an annual refresher became T3.4 • CR2.1 Use automated tools became CR1.4 • SE2.1 Use code protection became SE3.2 • SE3.1 Use code signing became SE2.4 • CR1.3 removed from the model
CHANGES FOR BSIMM1 (110 ACTIVITIES)	<ul style="list-style-type: none"> • Added 110 activities

TABLE 6. ACTIVITY CHANGES OVER TIME. This table allows for historical review of how BSIMM activities have been added, moved, and deleted since inception.

D. DATA: BSIMM13

Every organization wants to do software security more effectively and efficiently. You can use this information to understand what the BSIMM community is doing today and how those efforts have evolved over time, then plan your own SSI changes.

The BSIMM data yields very interesting analytical results, as shown throughout this document. Figure 17 shows the highest-resolution observation data that is published. Organizations can use this information to note how often we observe each activity across all 130 participants to help plan their next areas of focus. Activities that are broadly popular will likely benefit your organization as well.

In Figure 17, we also identified the most common activity in each practice (highlighted in orange). To provide some perspective on what “most common” means, although T1.1 is the most common activity in the Training practice with 71 observations, Table 7 shows that it isn’t in the top 20 activities across all the practices.

To provide another view into this data, we created a spider chart by noting the percentage of activities observed for each practice per BSIMM participant (normalized scale), then averaging these values over the group of 130 firms to produce 12 numbers (one for each practice). The resulting spider chart (Figure 15) plots these values on spokes corresponding to the 12 BSIMM practices. Note that performing a larger number of activities is often a sign of SSI maturity. Other interesting analyses are possible, of course, such as those at www.ieeexplore.ieee.org/document/9732894.

The range of observed scores in the current data pool is 12 for the lower score and 100 for the higher score, indicating a wide range of SSI maturity levels in the BSIMM13 data.

AGE-BASED PROGRAM CHANGES

Figure 16 shows the distribution of scores among the population of 130 participating firms. To create this graph, we divided the scores into six bins that are then further divided by the assessment iteration (round 1, round 2, and round 3+). As you can see, the scores represent a slightly skewed bell curve. We also plotted the average age of the firms’ SSIs in each bin as a horizontal line. In general, firms where more BSIMM activities were observed have older SSIs and are more likely to have performed multiple BSIMM measurements.

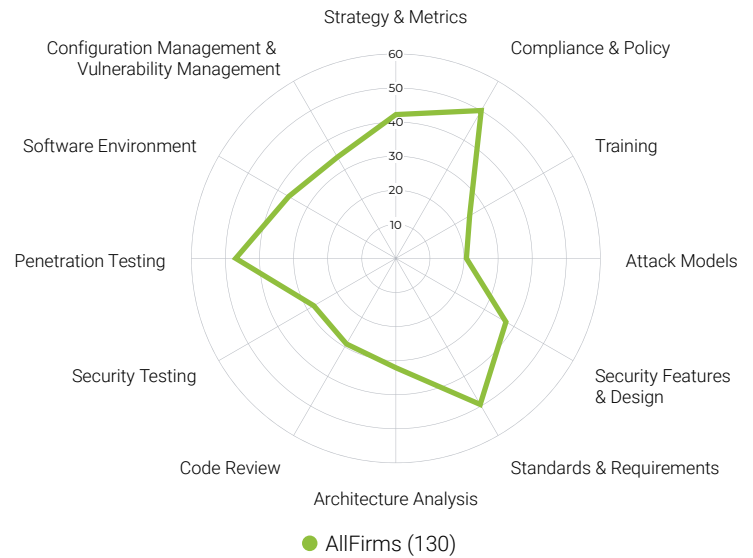


FIGURE 15. ALLFIRMS SPIDER CHART. This diagram shows the average percentage of normalized observations collectively reached in each practice by the 130 BSIMM13 firms. Across these firms, the normalized observations are higher in the Compliance & Policy, Standards & Requirements, and Penetration Testing practices compared to Training, Attack Models, and Security Testing.

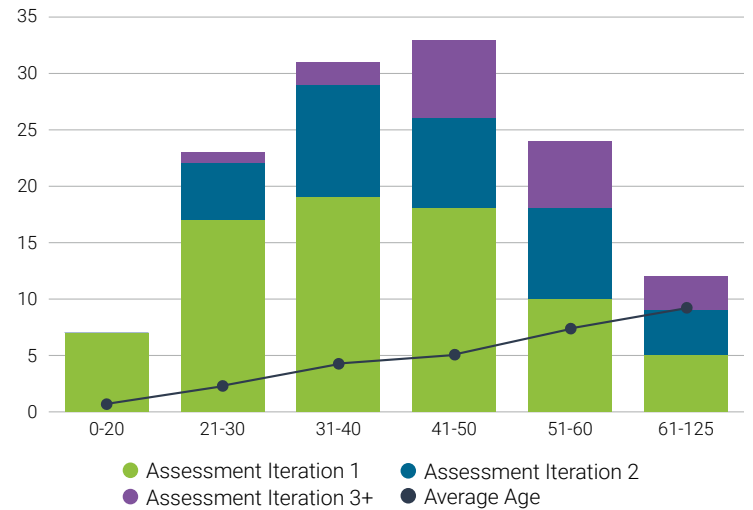


FIGURE 16. BSIMM SCORE DISTRIBUTION. Assessment scores most frequently fall into the 41 to 50 range in BSIMM13, versus 31 to 40 in BSIMM12 (not shown), with an average SSG age of 5.0 years. In general, firms that mature and continue to use the BSIMM as a measurement tool over time (e.g., round 2, round 3+), tend to have higher scores. Refer to Appendix F for more details on how SSIs evolve over multiple measurements.

GOVERNANCE			INTELLIGENCE			SSDL TOUCHPOINTS			DEPLOYMENT		
ACTIVITY	BSIMM13 FIRMS (OUT OF 130)	BSIMM13 FIRMS (PERCENTAGE)	ACTIVITY	BSIMM13 FIRMS (OUT OF 130)	BSIMM13 FIRMS (PERCENTAGE)	ACTIVITY	BSIMM13 FIRMS (OUT OF 130)	BSIMM13 FIRMS (PERCENTAGE)	ACTIVITY	BSIMM13 FIRMS (OUT OF 130)	BSIMM13 FIRMS (PERCENTAGE)
STRATEGY & METRICS			ATTACK MODELS			ARCHITECTURE ANALYSIS			PENETRATION TESTING		
[SM1.1]	98	75.4%	[AM1.2]	80	61.5%	[AA1.1]	113	86.9%	[PT1.1]	114	87.7%
[SM1.3]	82	63.1%	[AM1.3]	42	32.3%	[AA1.2]	53	40.8%	[PT1.2]	102	78.5%
[SM1.4]	117	90.0%	[AM1.5]	76	58.5%	[AA1.4]	69	53.1%	[PT1.3]	88	67.7%
[SM2.1]	73	56.2%	[AM2.1]	16	12.3%	[AA2.1]	31	23.9%	[PT2.2]	38	29.2%
[SM2.2]	63	48.5%	[AM2.2]	11	8.5%	[AA2.2]	32	24.6%	[PT2.3]	45	34.6%
[SM2.3]	69	53.1%	[AM2.5]	16	12.3%	[AA2.4]	38	29.2%	[PT3.1]	26	20.0%
[SM2.6]	71	54.6%	[AM2.6]	16	12.3%	[AA3.1]	20	15.4%	[PT3.2]	15	11.5%
[SM2.7]	64	49.2%	[AM2.7]	14	10.8%	[AA3.2]	4	3.1%			
[SM3.1]	27	20.8%	[AM3.1]	9	6.9%	[AA3.3]	15	11.5%			
[SM3.2]	18	13.9%	[AM3.2]	5	3.9%						
[SM3.3]	26	20.0%	[AM3.3]	11	8.5%						
[SM3.4]	5	3.9%									
[SM3.5]	0	0.0%									
COMPLIANCE & POLICY			SECURITY FEATURES & DESIGN			CODE REVIEW			SOFTWARE ENVIRONMENT		
[CP1.1]	101	77.7%	[SFD1.1]	104	80.0%	[CR1.2]	83	63.9%	[SE1.1]	87	66.9%
[CP1.2]	115	88.5%	[SFD1.2]	90	69.2%	[CR1.4]	107	82.3%	[SE1.2]	115	88.5%
[CP1.3]	98	75.4%	[SFD2.1]	39	30.0%	[CR1.5]	62	47.7%	[SE1.3]	79	60.8%
[CP2.1]	58	44.6%	[SFD2.2]	64	49.2%	[CR1.7]	54	41.5%	[SE2.2]	57	43.9%
[CP2.2]	59	45.4%	[SFD3.1]	17	13.1%	[CR2.6]	28	21.5%	[SE2.4]	39	30.0%
[CP2.3]	73	56.2%	[SFD3.2]	18	13.9%	[CR2.7]	20	15.4%	[SE2.5]	52	40.0%
[CP2.4]	62	47.7%	[SFD3.3]	7	5.4%	[CR2.8]	34	26.2%	[SE2.7]	42	32.3%
[CP2.5]	82	63.1%				[CR3.2]	14	10.8%	[SE3.2]	19	14.6%
[CP3.1]	30	23.1%				[CR3.3]	8	6.2%	[SE3.3]	11	8.5%
[CP3.2]	28	21.5%				[CR3.4]	2	1.5%	[SE3.6]	18	13.9%
[CP3.3]	11	8.5%				[CR3.5]	3	2.3%	[SE3.8]	0	0.0%
TRAINING			STANDARDS & REQUIREMENTS			SECURITY TESTING			CONFIG. MGMT. & VULN. MGMT.		
[T1.1]	71	54.6%	[SR1.1]	96	73.9%	[ST1.1]	108	83.1%	[CMVM1.1]	114	87.7%
[T1.7]	58	44.6%	[SR1.2]	101	77.7%	[ST1.3]	97	74.6%	[CMVM1.2]	100	76.9%
[T1.8]	53	40.8%	[SR1.3]	103	79.2%	[ST1.4]	56	43.1%	[CMVM2.1]	95	73.1%
[T2.5]	38	29.2%	[SR2.2]	80	61.5%	[ST2.4]	25	19.2%	[CMVM2.2]	98	75.4%
[T2.8]	28	21.5%	[SR2.4]	92	70.8%	[ST2.5]	31	23.9%	[CMVM2.3]	62	47.7%
[T2.9]	33	25.4%	[SR2.5]	63	48.5%	[ST2.6]	21	16.2%	[CMVM3.1]	11	8.5%
[T2.10]	28	21.5%	[SR2.7]	53	40.8%	[ST3.3]	12	9.2%	[CMVM3.2]	19	14.6%
[T2.11]	27	20.8%	[SR3.2]	19	14.6%	[ST3.4]	4	3.1%	[CMVM3.3]	18	13.9%
[T3.1]	9	6.9%	[SR3.3]	17	13.1%	[ST3.5]	4	3.1%	[CMVM3.4]	26	20.0%
[T3.2]	16	12.3%	[SR3.4]	19	14.6%	[ST3.6]	3	2.3%	[CMVM3.5]	13	10.0%
[T3.5]	22	16.9%							[CMVM3.6]	3	2.3%
[T3.6]	7	5.4%							[CMVM3.7]	20	15.4%
									[CMVM3.8]	0	0.0%

FIGURE 17. BSIMM13 SCORECARD. This scorecard shows how often we observed each of the BSIMM13 activities in the data pool of 130 firms.

ACTIVITY CHANGES OVER TIME

The popular business book, *The 7 Habits of Highly Effective People*, explores the theory that successful individuals share common qualities in achieving their goals, and that these qualities can be identified and applied by others. The same premise can also be applied to SSIs. Table 7 lists the 20 most observed activities in the BSIMM13 data pool. The data suggests that if your organization is working on its own SSI, you should consider implementing these activities. As a reminder of how practices and activity labeling works, activity SM1.4 is from the Strategy & Metrics practice, and it was observed in 90% of the 130 BSIMM13 participant organizations.

Instead of the top 20 activities overall, Table 8 shows the most common activity in each BSIMM practice (e.g., SM1.4 refers to an activity in the Strategy & Metrics practice). Although we can't directly conclude that these 12 activities are necessary for all SSIs, we can say with confidence that they're commonly found in initiatives whose efforts span all 12 practices. This suggests that if an organization is working on an initiative of its own, their efforts will likely include the majority of these 12 activities over time.

In addition to looking at the most common activities, we can also analyze the fastest-growing activity observation rates between BSIMM12 and BSIMM13. In BSIMM12, we observed 25 activities that were already very common, having more than 80 observations each. Table 9 shows that even among these common activities, some experienced even more above average growth.

BSIMM13 TOP ACTIVITIES BY PRACTICE		
ACTIVITY	PERCENTAGE	DESCRIPTION
[SM1.4]	90.0%	Implement security checkpoints and associated governance.
[CP1.2]	88.5%	Identify privacy obligations.
[T1.1]	54.6%	Conduct software security awareness training.
[AM1.2]	61.5%	Use a data classification scheme for software inventory.
[SFD1.1]	80.0%	Integrate and deliver security features.
[SR1.3]	79.2%	Translate compliance constraints to requirements.
[AA1.1]	86.9%	Perform security feature review.
[CR1.4]	82.3%	Use automated code review tools.
[ST1.1]	83.1%	Perform edge/boundary value condition testing during QA.
[PT1.1]	87.7%	Use external penetration testers to find problems.
[SE1.2]	88.5%	Ensure host and network security basics are in place.
[CMVM1.1]	87.7%	Create or interface with incident response.

TABLE 8. MOST COMMON ACTIVITY PER PRACTICE. This table shows the most observed activity in each of the 12 BSIMM practices for the entire data pool of 130 participant firms.

BSIMM13 TOP 20 ACTIVITIES BY OBSERVATION PERCENTAGE		
ACTIVITY	PERCENTAGE	DESCRIPTION
[SM1.4]	90.0%	Implement security checkpoints and associated governance.
[SE1.2]	88.5%	Ensure host and network security basics are in place.
[CP1.2]	88.5%	Identify privacy obligations.
[CMVM1.1]	87.7%	Create or interface with incident response.
[PT1.1]	87.7%	Use external penetration testers to find problems.
[AA1.1]	86.9%	Perform security feature review.
[ST1.1]	83.1%	Perform edge/boundary value condition testing during QA.
[CR1.4]	82.3%	Use automated code review tools.
[SFD1.1]	80.0%	Integrate and deliver security features.
[SR1.3]	79.2%	Translate compliance constraints to requirements.
[PT1.2]	78.5%	Feed results to the defect management and mitigation system.
[SR1.2]	77.7%	Create a security portal.
[CP1.1]	77.7%	Unify regulatory pressures.
[CMVM2.2]	76.9%	Identify software defects found in operations monitoring and feed them back to development.
[SM1.1]	75.4%	Publish process and evolve as necessary.
[CP1.3]	75.4%	Create policy.
[CMVM2.2]	75.4%	Track software bugs found in operations through the fix process.
[ST1.3]	74.6%	Drive tests with security requirements and security features.
[SR1.1]	73.8%	Create security standards.
[CMVM2.1]	73.1%	Have emergency response.

TABLE 7. TOP 20 ACTIVITIES BY OBSERVATION PERCENTAGE. Shown here are the most observed activities in the BSIMM13 data pool of 130 firms. This frequent observation means that each activity has broad applicability across a wide variety of SSIs.

Tables 7 and 9 can help you understand what most firms are already doing and discover potential gaps in your program. Another way to look at the growth of activities between BSIMM12 and BSIMM13 is to look for trends, such as a high growth in observation rates among common controls. There were 29 activities in BSIMM12 with observations in the range of 40 to 79. The observation rate for six of these activities, shown in Table 10, grew at 20% or higher. In addition, there were 24 activities with observations in the range 20 to 39, and six of them grew at 25% or more (see Table 11).

If we analyze these fast-growing activities, we observe a few areas of interest to consider in your SSI:

- Now that [CR1.4 Use automated code review tools] is observed in more than 82% of all firms, SSGs are starting to enforce code reviews for all projects [CR1.5]. In addition, firms are starting to scale their security testing across their complete application portfolio [PT2.3] and are expanding beyond doing DAST to include security testing in QA automation [ST2.5]. This might highlight that more firms are moving to the maturing phase of their SSIs (see Appendix B) and are now working on the scalability, efficiency, and effectiveness aspects of their programs.
- Firms have already invested heavily in fundamental activities to manage their compliance obligations [CP1.1 Unify regulatory pressure] and [SR1.3 Translate compliance constraints to requirements], both of which are found in Table 7. In addition, firms are increasing their efforts to manage compliance risk [CP2.2] and creating a repeatable way to document their compliance story [CP3.1]. These are potentially additional examples of what organizations do once they enter the maturing phase of their SSIs.
- In response to multiple high-profile breaches in the last few years, we observed significant growth in activities to address software supply chain risk management (SSCRM) (see Trends and Insights). Potentially, organizations are also responding to these breaches by investing in attack intelligence [AM1.5] they can use to improve their programs.

BSIMM13 HIGH-GROWTH ACTIVITIES (1)		
ACTIVITY	GROWTH	DESCRIPTION
[SR1.2]	14.8%	Create a security portal.
[ST1.3]	11.5%	Drive tests with security requirements and security features.
[CP1.3]	11.4%	Create policy.

TABLE 9. VERY COMMON ACTIVITIES WITH ABOVE AVERAGE GROWTH. This table shows that firms, including those just starting their SSIs, continue to invest into fundamental activities.

BSIMM13 HIGH-GROWTH ACTIVITIES (2)		
ACTIVITY	GROWTH	DESCRIPTION
[SE1.3]	33.9%	Implement cloud security controls.
[CR1.5]	26.5%	Make code review mandatory for all projects.
[SR2.2]	25.0%	Create a standards review process.
[AM1.5]	24.6%	Gather and use attack intelligence.
[SR2.4]	24.3%	Identify open source.
[CP2.2]	20.4%	Require security sign-off for compliance-related risk.

TABLE 10. COMMON ACTIVITIES WITH HIGH GROWTH IN OBSERVATION RATES. This table shows an ongoing trend of investment in common activities. If you are not performing or planning to perform these activities, consider them during your next planning cycle.

BSIMM13 HIGH-GROWTH ACTIVITIES (3)		
ACTIVITY	GROWTH	DESCRIPTION
[SR2.7]	51.4%	Control open source risk.
[ST2.5]	47.6%	Include security tests in QA automation.
[PT2.3]	32.4%	Schedule periodic penetration tests for application coverage.
[CMVM3.4]	30.0%	Operate a bug bounty program.
[SE2.7]	27.3%	Use orchestration for containers and virtualized environments.
[CP3.1]	25.0%	Document a software compliance story.

TABLE 11. ACTIVITIES WITH HIGH GROWTH IN OBSERVATION RATES. This table shows potential new trends in the BSIMM13 data pool.

E. DATA ANALYSIS: VERTICALS

- While every company is a software company these days, there are differences in SSI implementation. You can use this information on how vertical markets approach software security to inform your own strategy.

An important use of the BSIMM data is to help everyone see how different groups of organizations approach the implementation of software security activities. Do certain groups focus more on governance than testing? Or perhaps architecture and secure-by-design components versus operational maintenance? What about training? Or vendor management? While it seems true that “every

company is becoming a software company,” different verticals still have their own priorities. The BSIMM data helps us to observe and analyze this.

An important use of the BSIMM data helps everyone see how different organizations approach implementing software security activities.

Table 12 shows how the representation of different verticals has grown and evolved over the history of the BSIMM. Financial, ISV, and technology firms were early adopters of the BSIMM, and we’ve recently seen increased participation by cloud firms.

BSIMM VERTICAL PARTICIPANTS OVER TIME								
	FINANCIAL	FINTECH	ISV	TECH	HEALTHCARE	INTERNET OF THINGS	CLOUD	INSURANCE
BSIMM13	44	15	38	33	11	19	35	15
BSIMM12	38	21	42	28	14	18	26	13
BSIMM11	42	21	46	27	14	17	30	14
BSIMM10	57		43	20	16	13	20	11
BSIMM9	50		42	22	19	16	17	10
BSIMM8	47		38	16	17	12	16	11
BSIMM7	42		30	14	15	12	15	10
BSIMM6	33		27	17	10	13		
BSIMM-V	26		25	14				
BSIMM4	19		19	13				
BSIMM3	17		15	10				
BSIMM2	12		7	7				
BSIMM1	4		4	2				

TABLE 12. BSIMM VERTICALS OVER TIME. The BSIMM community has grown over the years as shown by growth in vertical representation. Remember that a firm can appear in more than one vertical. Note also that FinTech became a separate vertical from Financial in BSIMM11.

IOT, CLOUD, AND ISV VERTICALS

IoT, cloud, and ISV firms each create software solutions, although these verticals usually deploy their solutions in different ways. Relative to BSIMM activities, cloud and ISV firms share a similar observation pattern, except for the Compliance & Policy and Architecture Analysis practices, where the ISV vertical is ahead of the Cloud vertical (see Figure 18). This might reflect the different relationships that ISVS and cloud firms have with their respective customers and perhaps the level of regulation and transparency required.

Using the vertical scorecards found later in this section (Figure 23), we can perform further analysis on similarities and differences between verticals. For example, we see that the observations putting ISVS ahead of the Cloud vertical in the Architecture Analysis practice are [AA1.2 Perform design review for high-risk applications] and [AA2.1 Perform architecture analysis using a defined process], where the observation rate for ISVS is almost 50% higher than the observation rate for Cloud. This difference indicates that ISVS spend significantly more effort on going beyond threat modeling [AA1.1] to perform design reviews and architecture analysis.

IoT firms exhibit a similar pattern when compared to the weighted average of the ISV and Cloud verticals, with a notably higher score in Architecture Analysis and a lower score in Penetration Testing (Figure 19). One potential explanation is that IoT manufacturers have less control of the production environments where their products are deployed, and their products are more likely to go for extended periods without software updates, which might reduce the perceived value of extended penetration testing and increase the perceived value of robust security designs. Similarly, it could be the case that IoT devices typically present an attack surface that's very different compared to a typical web application, and IoT devices usually aren't sitting in front of large databases of PII or other private information.

FINANCIAL, HEALTHCARE, AND INSURANCE VERTICALS

Three verticals in the BSIMM operate in highly regulated industries: Financial, Healthcare, and Insurance (see Figure 20). In our long experience with the BSIMM, we've seen large financial firms reacting to regulatory pressures by starting SSIs earlier than insurance and healthcare firms. However, for the first time, the SSG average ages for financial services and insurance firms are now the same, at 5.2 years, compared to 4.5 years in healthcare firms. Despite the narrowing of this age difference, financial firms still display higher maturity. This likely reflects a longer history of software security activity in the Financial vertical, coupled with an influx of younger financial firms that have comparatively new but relatively mature SSGs.

Although organizations in the Healthcare vertical include some mature outliers, the data for these three regulated verticals shows it lags the others in most practices but is ahead in Architecture Analysis. Compared to financial firms, we see a similar picture in the Insurance vertical, which is ahead in Security Testing but close or lagging in other practices. The biggest differences between the Insurance and Financial verticals are in Compliance & Policy, Security Features & Design, Penetration Testing, and Configuration Management & Vulnerability Management, where the Financial vertical leads Insurance.

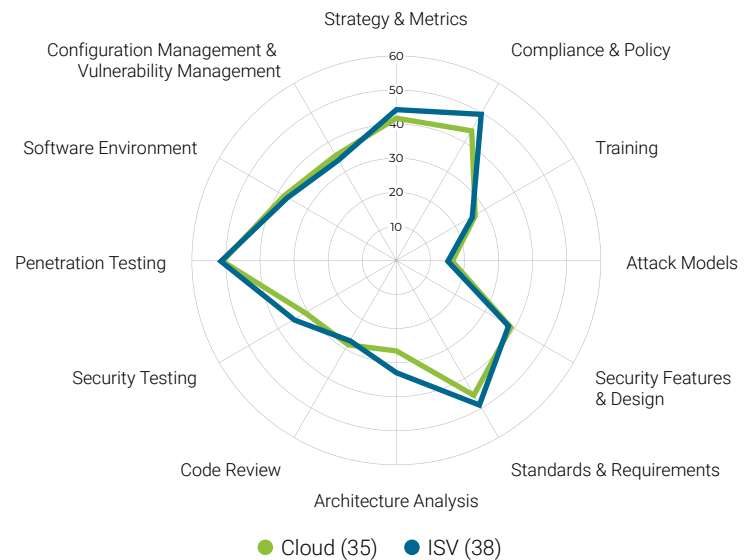


FIGURE 18. COMPARING CLOUD AND ISV VERTICALS. This diagram helps explain the differences, on a percentage scale, between practices in the Cloud and ISV verticals. Here, we see noticeable differences in the Compliance & Policy and Architecture Analysis practices.

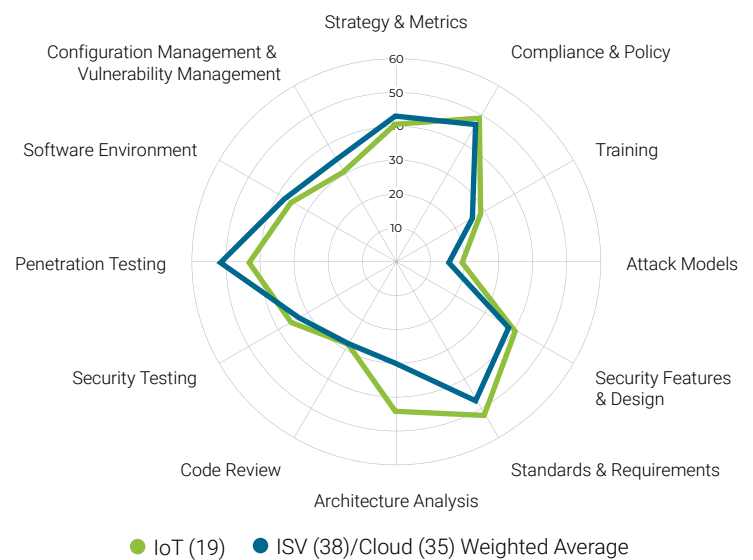


FIGURE 19. COMPARING IOT AND THE WEIGHTED AVERAGE OF ISV AND CLOUD. While the ISV and Cloud verticals are very similar, there are significant variations between IoT and those two verticals. The differences, on a percentage scale, in risk and deployment models, along with customer expectations, can explain the distinctions in their SSIs.

FINANCIAL AND TECHNOLOGY VERTICALS

Financial and Technology are the two verticals with the highest BSIMM scores. Figure 21 shows that while the average score across both verticals is similar in most practices, there are significant differences as well. Financial firms have a higher average score in Compliance & Policy, likely due to more stringent regulatory requirements. Technology firms have a higher average score in Architecture Analysis and Security Testing.

To understand more about the differences in these two practices, we analyzed the vertical scorecards found later in this section (Figure 23). In the Architecture Analysis practice, while financial firms have a high observation rate for threat modeling [AA1.1 Perform security feature review], the observation rates for design review [AA1.2 Perform design review for high-risk applications] and architecture risk analysis [AA2.1 Perform architecture analysis using a defined process] are almost three times higher in the Technology vertical compared to the Financial one. In addition, the observation rates for enabling engineering teams to be self-sufficient in performing architecture analysis ([AA3.1 Have engineering teams lead AA process] and [AA3.3 Make the SSG available as an AA resource or mentor]) are more than six times higher among technology firms compared to financial firms.

One explanation for this difference is the tighter relationship between hardware and software in many technology products. When the software must be closely mated to its hardware, then architecture analysis and engineering-driven design reviews are much more important to long-term success for products in the field. This trend seems to hold for IoT firms, and perhaps even for healthcare firms that are making IoT devices, which are doing more in the Architecture Analysis practice as compared to the overall data pool.

In the Security Testing practice, we see significantly higher observation rates for technology firms even when we ignore [ST2.6 Perform fuzz testing customized to application APIs], where we expect technology firms to perform a lot more fuzzing compared to financial ones. This includes fundamental activities such as [ST1.1 Perform edge/boundary value condition testing during QA] and [ST1.3 Drive tests with security requirements and security features]. When it comes to automation of security testing ([ST1.4 Integrate opaque-box security tools into the QA process], [ST2.5 Drive QA tests with AST results], and [ST3.4 Leverage code coverage analysis]), the observation rate for technology firms is almost double that of financial firms. The difference is even more pronounced when we look at activities [ST2.4 Drive QA tests with AST results] and [ST3.5 Begin to build and apply adversarial security tests (abuse cases)], which enable more in-depth testing. For these activities, the observation rate for technology firms is five times higher that it is for financial ones.

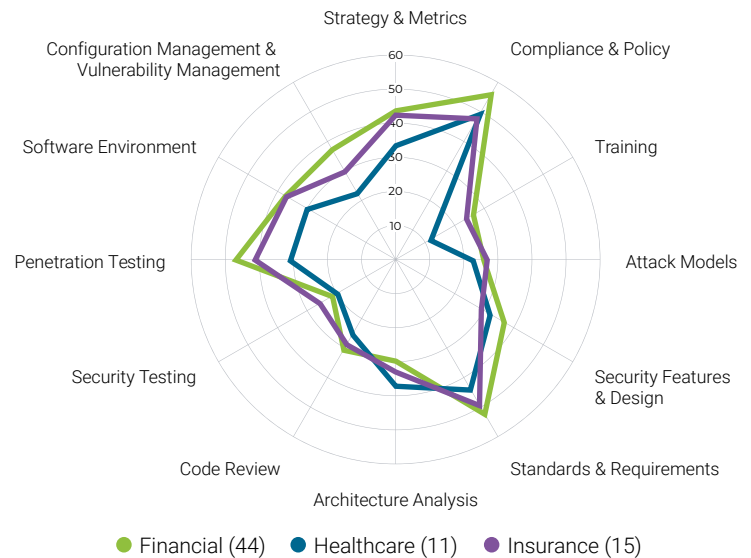


FIGURE 20. FINANCIAL VS. HEALTHCARE VS. INSURANCE. Even verticals that are similarly highly regulated exhibit significant differences in their SSIs. While they all have a focus on Compliance & Policy, there are significant differences, on a percentage scale, in most other practices, indicating that each vertical is responding to its regulatory obligations in its own way.



FIGURE 21. FINANCIAL VS. TECHNOLOGY. Technology firms appear to invest significantly more effort into in-depth design reviews, automation of security testing, and enablement of engineering teams to be self-sufficient, resulting in the differences, on a percentage scale, seen above. One potential explanation is that many technology firms build long-life products that they ship to customers and therefore perform more in-depth analysis before release.

TECHNOLOGY VS. NON-TECHNOLOGY

The Technology vertical stands out as the one with the least similarity to the other verticals. As such, it's informative to make a comparison between technology firms and everyone else, as illustrated in Figure 22. The biggest differences where technology firms lead everyone else are in Architecture Analysis and Security Testing, which could be indicative of a comparatively higher level of engineering rigor. The only practice where technology firms trail everyone else is Compliance & Policy, which reflects that other verticals are more closely regulated than technology firms.



FIGURE 22. TECHNOLOGY VS. NON-TECHNOLOGY. Shown here is a comparison of the Technology vertical versus the rest of the data pool, shown on a percentage scale. The average SSG age for technology firms is 5.9 years vs. 4.7 for all other firms, which could be one of the reasons for their higher maturity.

VERTICAL SCORECARDS

Figure 23 shows the BSIMM scorecards for the eight verticals compared side by side, allowing for discovery of differences and similarities between verticals. This report includes some new information for the vertical scorecards:

- For each activity per vertical, we are presenting the observation rate as a percentage (e.g., 74% of firms in the Cloud vertical are performing [SM1.1]).
- To show the biggest outliers within each vertical, we highlighted activities where observation rates are either at least 1.75 standard deviations above average (highlighted in blue) or at least 1.75 standard deviations below average (highlighted in dark orange). Use these highlighted differences to identify apparently higher- and lower-value activities unique to a vertical.
- We also highlighted five activities (see the activity column) with the least differences between verticals (light orange color) and five activities with the largest differences between verticals (teal color).

The activities in light orange appear to be uniformly applicable across all verticals, while those in teal appear to be more vertical-specific.

- We excluded in our analysis the activities with low observation rates (lower than 10 for all firms in the data pool) for bullets #2 and #3 above.

The following are observations from Figure 23:

- The five activities with the least variation in observation rate between verticals, not surprisingly, are some of the most common activities in BSIMM13. These are [SM1.4 Implement security checkpoints and associated governance], [SR1.3 Translate compliance constraints to requirements], [ST1.1 Perform edge/boundary value condition testing during QA], [SE1.2 Ensure host and network security basics are in place], and [CMVM1.1 Create or interface with incident response]. This is another indicator that these activities are applicable to all SSIs, independent of what vertical the firm is in.
- Activity [AM3.3 Monitor automated asset creation] was introduced in BSIMM10. It has one of the largest differences between verticals, and its observation rate for the Financial vertical is significantly above the overall average. This is an indication that financial firms are early adopters of [AM3.3] and the leaders in implementing this activity. In addition, the observation rate for [CMVM3.5 Automate verification of operational infrastructure security] (also introduced in BSIMM10) among financial firms is also significantly above the average. This is another indicator that financial firms are early adopters as well as potential leaders in the shift everywhere approach.
- Another three activities with large differences in observation rates between verticals are [ST2.6 Perform fuzz testing customized to application APIs], [ST3.3 Drive tests with design review results], and [SE3.2 Use code protection]. For these activities, the observation rate for technology firms is significantly higher than the average, an indication that some verticals potentially focus on specific activities because of their unique technology stacks (e.g., very API driven) and because they publish their software across trust boundaries (e.g., shipping products to customers).
- The Healthcare vertical (11 firms) has a significantly lower average BSIMM score (37.7) versus the remaining 119 BSIMM13 firms (45.0), so it is not surprising that the Healthcare vertical has the most observation rates highlighted in orange compared to other verticals. This gap is especially evident when we look at developer enablement—the Training practice overall and [CR1.7 Assign code review tool mentors] in particular. At the same time, Healthcare is leading all verticals in unifying regulatory pressures [CP1.1].
- FinTech has significantly higher observation rates for [SR2.7 Control open source risk] and [CR2.6 Use custom rules with automated code review tools] compared to other verticals. It's unclear whether there is a correlation between these two activities; for example, are FinTech firms leveraging custom SAST rules to enforce their open source policy and manage the risk?

GOVERNANCE

ACTIVITY	CLOUD (OF 35)	FINANCIAL (OF 44)	FINTECH (OF 15)	HEALTHCARE (OF 11)	INSURANCE (OF 15)	IOT (OF 19)	ISV (OF 38)	TECH (OF 33)
STRATEGY & METRICS								
[SM1.1]	74%	73%	67%	82%	87%	95%	79%	94%
[SM1.3]	69%	66%	60%	45%	67%	47%	71%	61%
[SM1.4]	86%	100%	87%	91%	93%	95%	84%	91%
[SM2.1]	66%	70%	80%	36%	73%	32%	50%	52%
[SM2.2]	46%	59%	47%	18%	33%	47%	50%	55%
[SM2.3]	63%	34%	60%	45%	47%	63%	76%	58%
[SM2.6]	49%	64%	47%	27%	47%	58%	55%	64%
[SM2.7]	46%	43%	53%	64%	53%	47%	50%	61%
[SM3.1]	23%	23%	33%	0%	7%	21%	26%	24%
[SM3.2]	9%	11%	13%	9%	13%	16%	16%	15%
[SM3.3]	17%	27%	20%	18%	27%	11%	18%	18%
[SM3.4]	0%	7%	7%	9%	13%	0%	3%	0%
[SM3.5]	0%	0%	0%	0%	0%	0%	0%	0%
COMPLIANCE & POLICY								
[CP1.1]	66%	82%	87%	100%	73%	79%	76%	73%
[CP1.2]	86%	95%	100%	100%	100%	95%	82%	82%
[CP1.3]	66%	84%	73%	73%	73%	79%	74%	76%
[CP2.1]	43%	50%	73%	45%	27%	47%	45%	36%
[CP2.2]	29%	55%	27%	36%	47%	68%	42%	58%
[CP2.3]	46%	64%	60%	64%	47%	58%	58%	64%
[CP2.4]	46%	57%	40%	36%	60%	26%	53%	42%
[CP2.5]	66%	68%	67%	55%	47%	53%	71%	52%
[CP3.1]	11%	32%	40%	18%	40%	11%	13%	15%
[CP3.2]	11%	30%	7%	27%	20%	16%	18%	24%
[CP3.3]	17%	9%	13%	0%	0%	11%	16%	12%
TRAINING								
[T1.1]	60%	55%	67%	18%	33%	74%	58%	61%
[T1.7]	51%	50%	40%	18%	47%	53%	45%	55%
[T1.8]	40%	55%	47%	27%	53%	21%	39%	33%
[T2.5]	34%	14%	40%	18%	33%	32%	45%	39%
[T2.8]	31%	9%	7%	9%	13%	42%	29%	39%
[T2.9]	17%	32%	7%	18%	27%	42%	16%	39%
[T2.10]	17%	27%	20%	9%	20%	26%	24%	24%
[T2.11]	17%	27%	7%	18%	27%	21%	16%	24%
[T3.1]	9%	7%	0%	0%	13%	5%	8%	15%
[T3.2]	17%	18%	20%	9%	13%	16%	11%	9%
[T3.5]	17%	23%	20%	0%	13%	11%	13%	21%
[T3.6]	9%	5%	7%	0%	0%	5%	5%	9%

INTELLIGENCE

ACTIVITY	CLOUD (OF 35)	FINANCIAL (OF 44)	FINTECH (OF 15)	HEALTHCARE (OF 11)	INSURANCE (OF 15)	IOT (OF 19)	ISV (OF 38)	TECH (OF 33)
ATTACK MODELS								
[AM1.2]	43%	86%	80%	91%	87%	32%	42%	42%
[AM1.3]	17%	43%	33%	45%	67%	26%	18%	24%
[AM1.5]	46%	73%	67%	73%	67%	58%	39%	58%
[AM2.1]	9%	16%	20%	9%	27%	16%	5%	12%
[AM2.2]	6%	9%	13%	9%	7%	16%	5%	15%
[AM2.5]	11%	11%	13%	9%	20%	21%	13%	24%
[AM2.6]	17%	7%	13%	0%	0%	16%	16%	30%
[AM2.7]	14%	14%	7%	9%	13%	11%	11%	9%
[AM3.1]	9%	7%	0%	0%	0%	11%	11%	9%
[AM3.2]	3%	5%	0%	9%	7%	5%	0%	3%
[AM3.3]	6%	18%	7%	0%	7%	5%	5%	0%
SECURITY FEATURES & DESIGN								
[SFD1.1]	77%	84%	93%	82%	87%	58%	76%	79%
[SFD1.2]	80%	64%	60%	64%	60%	84%	82%	79%
[SFD2.1]	31%	30%	47%	18%	7%	37%	32%	39%
[SFD2.2]	60%	39%	40%	36%	33%	74%	58%	67%
[SFD3.1]	3%	25%	7%	18%	13%	11%	3%	18%
[SFD3.2]	17%	14%	13%	9%	7%	11%	13%	12%
[SFD3.3]	3%	7%	0%	0%	0%	11%	3%	12%
STANDARDS & REQUIREMENTS								
[SR1.1]	60%	82%	80%	73%	87%	79%	63%	73%
[SR1.2]	83%	75%	67%	64%	73%	84%	84%	88%
[SR1.3]	69%	84%	80%	82%	73%	84%	82%	79%
[SR2.2]	51%	75%	60%	55%	87%	58%	45%	61%
[SR2.4]	63%	77%	73%	73%	73%	79%	74%	79%
[SR2.5]	43%	55%	47%	45%	47%	42%	50%	52%
[SR2.7]	43%	43%	67%	27%	33%	26%	45%	45%
v	6%	14%	7%	27%	20%	21%	16%	15%
[SR3.3]	17%	7%	20%	0%	0%	21%	11%	27%
[SR3.4]	20%	18%	7%	0%	7%	26%	18%	15%

SSDL TOUCHPOINTS

ACTIVITY	CLOUD (OF 35)	FINANCIAL (OF 44)	FINTECH (OF 15)	HEALTHCARE (OF 11)	INSURANCE (OF 15)	IOT (OF 19)	ISV (OF 38)	TECH (OF 33)
ARCHITECTURE ANALYSIS								
[AA1.1]	91%	89%	100%	73%	100%	84%	92%	85%
[AA1.2]	31%	30%	20%	45%	27%	74%	45%	70%
[AA1.4]	31%	82%	73%	64%	80%	21%	32%	27%
[AA2.1]	20%	16%	0%	36%	20%	47%	32%	52%
[AA2.2]	20%	14%	0%	36%	27%	53%	29%	58%
[AA2.4]	17%	25%	20%	45%	27%	47%	32%	39%
[AA3.1]	14%	5%	0%	18%	13%	42%	21%	45%
[AA3.2]	0%	5%	0%	9%	0%	5%	0%	6%
[AA3.3]	11%	7%	0%	9%	7%	21%	13%	24%
CODE REVIEW								
[CR1.2]	66%	68%	53%	55%	60%	79%	63%	61%
[CR1.4]	77%	86%	93%	82%	93%	79%	82%	79%
[CR1.5]	40%	48%	47%	55%	40%	58%	47%	58%
[CR1.7]	43%	39%	53%	27%	53%	37%	47%	45%
[CR2.6]	29%	20%	40%	18%	13%	16%	21%	21%
[CR2.7]	17%	18%	13%	9%	27%	16%	5%	24%
[CR2.8]	20%	36%	27%	27%	20%	11%	21%	24%
[CR3.2]	9%	16%	13%	0%	7%	5%	5%	12%
[CR3.3]	9%	5%	13%	9%	7%	5%	3%	6%
[CR3.4]	0%	2%	0%	0%	0%	0%	0%	3%
[CR3.5]	3%	2%	0%	0%	0%	0%	3%	3%
SECURITY TESTING								
[ST1.1]	89%	77%	87%	82%	80%	89%	92%	94%
[ST1.3]	77%	66%	73%	64%	87%	84%	89%	85%
[ST1.4]	37%	36%	60%	36%	47%	63%	50%	64%
[ST2.4]	23%	9%	27%	0%	7%	32%	24%	36%
[ST2.5]	31%	18%	27%	9%	20%	26%	37%	36%
[ST2.6]	20%	5%	7%	0%	0%	26%	29%	39%
[ST3.3]	11%	0%	0%	9%	7%	26%	13%	27%
[ST3.4]	6%	2%	0%	0%	7%	0%	3%	9%
[ST3.5]	6%	0%	0%	0%	0%	5%	5%	9%
[ST3.6]	6%	5%	7%	0%	7%	0%	3%	0%

DEPLOYMENT								
ACTIVITY	CLOUD (OF 35)	FINANCIAL (OF 44)	FINTECH (OF 15)	HEALTHCARE (OF 11)	INSURANCE (OF 15)	IOT (OF 19)	ISV (OF 38)	TECH (OF 33)
PENETRATION TESTING								
[PT1.1]	89%	86%	93%	91%	93%	74%	95%	88%
[PT1.2]	83%	75%	100%	64%	67%	58%	89%	76%
[PT1.3]	63%	75%	73%	64%	73%	58%	66%	61%
[PT2.2]	40%	23%	40%	0%	13%	42%	37%	39%
[PT2.3]	49%	45%	33%	18%	40%	21%	45%	21%
[PT3.1]	23%	18%	27%	9%	7%	32%	18%	33%
[PT3.2]	11%	11%	27%	0%	0%	16%	11%	21%
SOFTWARE ENVIRONMENT								
[SE1.1]	60%	91%	80%	82%	93%	37%	53%	45%
[SE1.2]	91%	93%	93%	91%	93%	89%	82%	94%
[SE1.3]	71%	70%	60%	64%	80%	37%	66%	42%
[SE2.2]	40%	45%	40%	9%	33%	63%	45%	61%
[SE2.4]	37%	9%	20%	9%	7%	74%	42%	70%
[SE2.5]	46%	41%	47%	36%	47%	37%	42%	42%
[SE2.7]	49%	34%	33%	27%	40%	5%	39%	18%
[SE3.2]	6%	7%	7%	0%	7%	26%	16%	39%
[SE3.3]	6%	11%	20%	18%	13%	0%	8%	3%
[SE3.6]	14%	16%	13%	0%	0%	21%	16%	21%
[SE3.8]	0%	0%	0%	0%	0%	0%	0%	0%
CONFIGURATION MANAGEMENT & VULNERABILITY MANAGEMENT								
[CMVM1.1]	83%	93%	93%	82%	93%	79%	87%	88%
[CMVM1.2]	86%	75%	93%	55%	73%	79%	84%	76%
[CMVM2.1]	74%	80%	80%	64%	80%	68%	79%	64%
[CMVM2.2]	80%	73%	80%	64%	60%	79%	84%	82%
[CMVM2.3]	43%	64%	47%	27%	40%	26%	37%	42%
[CMVM3.1]	9%	9%	13%	0%	0%	11%	8%	15%
[CMVM3.2]	11%	20%	13%	0%	0%	16%	13%	21%
[CMVM3.3]	17%	18%	13%	9%	20%	5%	8%	15%
[CMVM3.4]	26%	23%	33%	0%	20%	11%	18%	18%
[CMVM3.5]	11%	20%	13%	0%	7%	5%	3%	6%
[CMVM3.6]	3%	2%	0%	0%	0%	5%	3%	6%
[CMVM3.7]	23%	16%	0%	0%	0%	16%	21%	24%
[CMVM3.8]	0%	0%	0%	0%	0%	0%	0%	0%

FIGURE 23. VERTICAL COMPARISON SCORECARD. This table allows for easy comparisons of observation rates for the eight verticals tracked in BSIMM13. A light orange color in the Activity column shows the five activities with the least differences in observation rates between verticals, whereas a teal color shows the five activities with the most differences. Blue and dark orange in the remaining columns show observation rates that are significantly different from the average, either above or below.

F. DATA ANALYSIS: LONGITUDINAL

- Every SSI changes over time as technologies, attackers, attacks, budgets, and everything else also changes.
- You can use this information to see whether your SSI's trajectory is similar to that of other programs.

The BSIMM captures real-world data about how organizations approach software security across their portfolio. Given the BSIMM's longevity, this data provides a unique snapshot of how the community of SSIs has evolved over the past 14 years, as well as how individual programs have changed from assessment to assessment.

BUILDING A MODEL FOR SOFTWARE SECURITY

In the late 1990s, software security began to flourish as a discipline separate from computer and network security. Researchers began to put more emphasis on studying the ways in which a developer can contribute to or unintentionally undermine the security of an application and started asking some specific questions: What kinds of bugs and flaws lead to security problems? How can we identify these problems systematically?

Within a few years, there was an emerging consensus that building secure software required more than smart individuals toiling away on guidance and training. Getting security right, especially across a software portfolio, meant being directly involved in the software development process, guiding it even as the process evolves. Since then, practitioners have come to learn that process, testing, and developer tools alone are insufficient: software security encompasses business, social, and organizational aspects as well.

Table 13 shows how the BSIMM has grown over the years. (Recall that our data freshness constraints, introduced with BSIMM-V and later tightened, cause data from firms with aging measurements to be removed.) BSIMM13 describes the work of 11,850 SSG and satellite members (champions) working directly in software security, impacting the security efforts of almost 410,000 developers.

Fifty-four of the current participating firms have been through at least two assessments, allowing us to study how their initiatives changed over time. Across North America, EMEA, and APAC, 35 firms are on their second assessment, 11 firms are on their third assessment, five are on their fourth, and two are on their fifth assessment. One North America firm has undertaken its sixth assessment, continuing its use of the BSIMM as an SSI planning and management tool. Figure 24 shows these firms by percentages across three major BSIMM regions.

BSIMM ASSESSMENTS DONE OVER TIME						
	FIRMS	1ST MEASUREMENTS	2ND MEASUREMENTS	3RD MEASUREMENTS	4TH MEASUREMENTS	DATA POOL MEASUREMENTS
BSIMM13	130	76	35	11	8	314
BSIMM12	128	76	31	14	7	341
BSIMM11	130	77	32	12	9	357
BSIMM10	122	72	29	13	8	339
BSIMM9	120	78	22	13	7	320
BSIMM8	109	73	20	11	5	256
BSIMM7	95	65	15	13	2	237
BSIMM6	78	52	16	8	2	202
BSIMM-V	67	46	17	4	0	161
BSIMM4	51	38	12	1	0	95
BSIMM3	42	31	11	0	0	81
BSIMM2	30	30	0	0	0	49
BSIMM1	9	9	0	0	0	9

TABLE 13. BSIMM ASSESSMENTS DONE OVER TIME. The chart shows how the BSIMM study has grown over the years, including how some firms have received multiple measurements.

CHANGES BETWEEN FIRST AND SECOND ASSESSMENTS

Fifty-four of the 130 firms in BSIMM13 have been measured at least twice. On average, the time between first and second measurements for those 54 firms was 32.6 months. Although observations of individual activities among the 12 practices come and go (as shown in Figure 26), in general, remeasurement over time shows a clear trend of increased maturity. The raw score went up in 48 of the 54 firms and remained the same in two. Across all 54 firms, the score increased by an average of 12.5 (36.7%) from the first to second measurement. Simply put, SSIs mature over time.

As shown in Figure 26, firms moving from their first assessment to their second tend to invest in:

- Defining their program ([SM1.1 Publish process and evolve as necessary], [SM2.1 Publish data about software security internally and use it to drive change]), scaling the program using the satellite ([SM2.3 Create or grow a satellite (security champions)]), and evangelizing the secure SDLC as well.
- Defining and enforcing policy and standards ([CP1.3 Create policy], [SR2.2 Create a standards review process])
- Managing vendors through boilerplate security SLAs ([CP2.4 Include software security SLAs in all vendor contracts], [SR2.5 Create SLA boilerplate])
- Identifying open source components ([SR2.4 Identify open source])

Figure 25 shows the average normalized observation rate per practice for the 54 firms that have had a second assessment. Over the average of about 32 months between the two assessments, we see clear growth in every practice, especially in Strategy & Metrics, Compliance & Policy, and Standards & Requirements. The practices with the highest overall growth align with the individual activities identified in Figure 26. The changes indicate that firms feel prepared for their first assessment after focusing on foundational and technical activities such as training and testing but then expand into governance as they mature their SSIs.

There are two factors causing the numerical changes seen in the longitudinal scorecard (Figure 26, showing 54 BSIMM13 firms moving from their first to second assessments). The first factor is that more firms have now done their second assessment (adding firms to this group), and the second is that we drop old data (removing firms from this group). Grouped together, the two factors can cause a significant amount of change in the group of firms that have had a second assessment, even if the change isn't directly visible in the scorecard.

For example, [CP2.5 Ensure executive awareness of compliance and privacy obligations] was newly observed in six firms, but it was either no longer observed in five firms doing their second assessment or decreased due to data aging out, giving a total change of 1 (as shown in the scorecard). As another example, [CR1.2 Perform opportunistic code review] was newly observed in five firms, but was no longer observed or was part of aged-out data in another five firms, causing the observation rate to stay the same at 32.

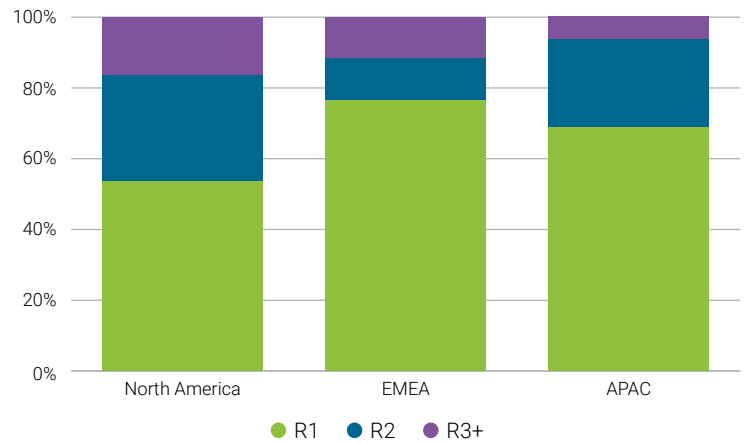


FIGURE 24. ONGOING USE OF THE BSIMM IN DRIVING ORGANIZATIONAL MATURITY. Organizations are continuing to do remeasurements to show that their efforts are achieving the desired results (e.g., about 55% of North America participants are on their first assessment).

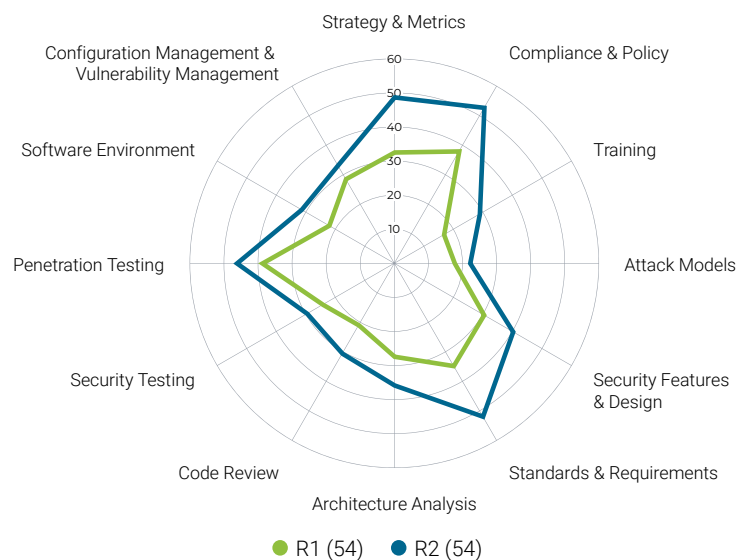


FIGURE 25. FIRMS ROUND 1 VS. FIRMS ROUND 2. This diagram illustrates the normalized observation rate change, on a percentage scale, in 54 firms between their first and second BSIMM assessments.

GOVERNANCE			INTELLIGENCE			SSDL TOUCHPOINTS			DEPLOYMENT		
ACTIVITY	BSIMM ROUND 1 (OF 54)	BSIMM ROUND 2 (OF 54)	ACTIVITY	BSIMM ROUND 1 (OF 54)	BSIMM ROUND 2 (OF 54)	ACTIVITY	BSIMM ROUND 1 (OF 54)	BSIMM ROUND 2 (OF 54)	ACTIVITY	BSIMM ROUND 1 (OF 54)	BSIMM ROUND 2 (OF 54)
STRATEGY & METRICS			ATTACK MODELS			ARCHITECTURE ANALYSIS			PENETRATION TESTING		
[SM1.1]	29	46	[AM1.2]	33	41	[AA1.1]	50	48	[PT1.1]	47	50
[SM1.3]	30	38	[AM1.3]	16	21	[AA1.2]	17	24	[PT1.2]	38	40
[SM1.4]	46	49	[AM1.5]	24	31	[AA1.4]	24	30	[PT1.3]	32	42
[SM2.1]	18	33	[AM2.1]	5	7	[AA2.1]	11	21	[PT2.2]	10	13
[SM2.2]	22	28	[AM2.2]	3	6	[AA2.2]	8	17	[PT2.3]	11	18
[SM2.3]	20	42	[AM2.5]	8	6	[AA2.4]	14	16	[PT3.1]	5	7
[SM2.6]	20	29	[AM2.6]	6	7	[AA3.1]	4	11	[PT3.2]	4	5
[SM2.7]	26	36	[AM2.7]	6	8	[AA3.2]	1	1			
[SM3.1]	11	16	[AM3.1]	2	2	[AA3.3]	4	6			
[SM3.2]	1	6	[AM3.2]	1	0						
[SM3.3]	6	17	[AM3.3]	1	3						
[SM3.4]	0	2									
[SM3.5]	0	0									
COMPLIANCE & POLICY			SECURITY FEATURES & DESIGN			CODE REVIEW			SOFTWARE ENVIRONMENT		
[CP1.1]	38	44	[SFD1.1]	40	45	[CR1.2]	32	32	[SE1.1]	23	36
[CP1.2]	44	49	[SFD1.2]	34	40	[CR1.4]	34	48	[SE1.2]	47	51
[CP1.3]	25	43	[SFD2.1]	12	21	[CR1.5]	16	26	[SE1.3]	3	21
[CP2.1]	15	29	[SFD2.2]	19	25	[CR1.7]	11	24	[SE2.2]	25	26
[CP2.2]	19	21	[SFD3.1]	4	10	[CR2.6]	7	12	[SE2.4]	13	17
[CP2.3]	20	27	[SFD3.2]	5	10	[CR2.7]	9	12	[SE2.5]	7	12
[CP2.4]	17	31	[SFD3.3]	1	1	[CR2.8]	15	20	[SE2.7]	3	8
[CP2.5]	29	30				[CR3.2]	1	6	[SE3.2]	7	5
[CP3.1]	9	16				[CR3.3]	0	1	[SE3.3]	3	5
[CP3.2]	9	17				[CR3.4]	0	0	[SE3.6]	1	6
[CP3.3]	1	6				[CR3.5]	0	1	[SE3.8]	0	0
TRAINING			STANDARDS & REQUIREMENTS			SECURITY TESTING			CONFIG. MGMT. & VULN. MGMT.		
[T1.1]	32	36	[SR1.1]	34	44	[ST1.1]	46	49	[CMVM1.1]	46	47
[T1.7]	18	31	[SR1.2]	34	45	[ST1.3]	43	39	[CMVM1.2]	45	39
[T1.8]	12	19	[SR1.3]	35	44	[ST1.4]	17	28	[CMVM2.1]	38	41
[T2.5]	11	19	[SR2.2]	18	34	[ST2.4]	5	11	[CMVM2.2]	37	40
[T2.8]	13	11	[SR2.4]	19	36	[ST2.5]	4	12	[CMVM2.3]	23	32
[T2.9]	9	19	[SR2.5]	14	30	[ST2.6]	11	10	[CMVM3.1]	1	2
[T2.10]	3	13	[SR2.7]	6	17	[ST3.3]	2	5	[CMVM3.2]	4	9
[T2.11]	2	15	[SR3.2]	8	11	[ST3.4]	1	2	[CMVM3.3]	3	6
[T3.1]	1	5	[SR3.3]	7	8	[ST3.5]	2	3	[CMVM3.4]	3	13
[T3.2]	5	8	[SR3.4]	13	11	[ST3.6]	0	1	[CMVM3.5]	1	0
[T3.5]	1	8							[CMVM3.6]	0	0
[T3.6]	2	3							[CMVM3.7]	0	4
									[CMVM3.8]	0	0

FIGURE 26. BSIMM13 REASSESSMENTS SCORECARD ROUND 1 Vs. ROUND 2. This chart shows how 54 SSIs changed between their first and second assessments. Dark orange shows the top five activities with the most increase in observations by count. Light orange shows the next five activities with the most increase in observations by count.

CHANGES BETWEEN FIRST AND THIRD ASSESSMENTS

Nineteen of the 130 firms in BSIMM13 have been measured at least three times. On average, the time between first and third measurements for those 19 firms was 56.9 months. Although individual activities among the 12 practices come and go (as shown on next page), in general, remeasurement over time shows a clear trend of increased maturity. The raw score went up in 18 of the 19 firms and decreased in one firm. Across all 19 firms, the score increased by an average of 20.1 (60.4%) from their first to their third measurements. Again, SSIs mature over time.

Although individual activities in the 12 practices come and go, in general, remeasurement over time shows a clear trend of increased maturity.

As shown in Figure 28, firms that move from their first assessment to their third over the course of about 56.9 months, in addition to changes shown previously, tend to invest in:

- Enabling self-sufficient engineering teams by leveraging investments in training ([T1.7 Deliver on-demand individual training], [T1.8 Include security resources in onboarding], [T2.9 Deliver role-specific advanced curriculum]), and static analysis tool mentors ([CR1.7])
- Securing cloud environments ([SE1.3])
- Identifying potential attackers ([AM1.3])

Interestingly, while Figure 27 shows growth in every practice, it shows only a slight increase in the Security Testing and Configuration Management & Vulnerability Management practices.

This could mean that most organizations do a variety of Security Testing and Configuration Management & Vulnerability Management activities earlier on in their journeys.

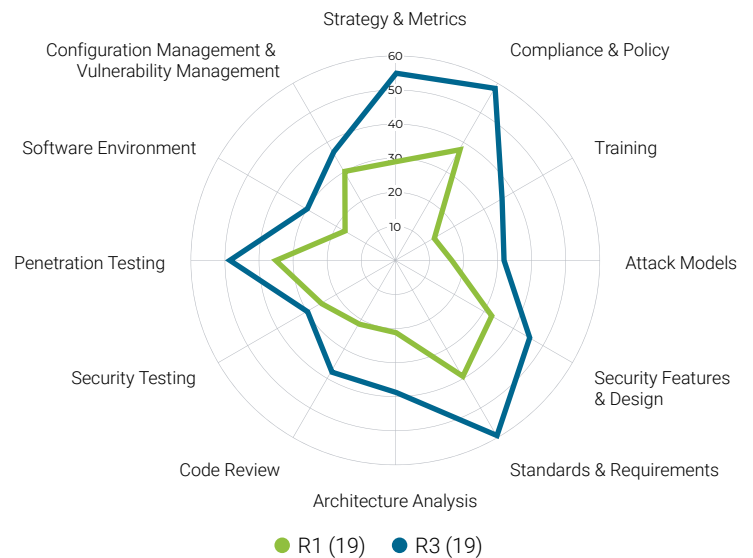


FIGURE 27. FIRMS ROUND 1 VS. FIRMS ROUND 3 SPIDER CHART. This diagram illustrates the normalized observation rate change, on a percentage scale, in 19 firms between their first and third BSIMM assessments.

GOVERNANCE			INTELLIGENCE			SSDL TOUCHPOINTS			DEPLOYMENT		
ACTIVITY	BSIMM ROUND 1 (OF 19)	BSIMM ROUND 3 (OF 19)	ACTIVITY	BSIMM ROUND 1 (OF 19)	BSIMM ROUND 3 (OF 19)	ACTIVITY	BSIMM ROUND 1 (OF 19)	BSIMM ROUND 3 (OF 19)	ACTIVITY	BSIMM ROUND 1 (OF 19)	BSIMM ROUND 3 (OF 19)
STRATEGY & METRICS			ATTACK MODELS			ARCHITECTURE ANALYSIS			PENETRATION TESTING		
[SM1.1]	7	19	[AM1.2]	12	17	[AA1.1]	16	19	[PT1.1]	17	18
[SM1.3]	9	15	[AM1.3]	4	14	[AA1.2]	4	9	[PT1.2]	10	17
[SM1.4]	17	18	[AM1.5]	8	11	[AA1.4]	9	14	[PT1.3]	9	13
[SM2.1]	6	15	[AM2.1]	3	8	[AA2.1]	2	7	[PT2.2]	3	7
[SM2.2]	6	11	[AM2.2]	1	4	[AA2.2]	0	5	[PT2.3]	6	5
[SM2.3]	8	13	[AM2.5]	4	5	[AA2.4]	3	8	[PT3.1]	2	3
[SM2.6]	6	12	[AM2.6]	1	2	[AA3.1]	1	3	[PT3.2]	1	3
[SM2.7]	6	15	[AM2.7]	2	3	[AA3.2]	0	0			
[SM3.1]	5	7	[AM3.1]	0	1	[AA3.3]	2	2			
[SM3.2]	0	6	[AM3.2]	0	1						
[SM3.3]	3	5	[AM3.3]	0	1						
[SM3.4]	0	1									
[SM3.5]	0	0									
COMPLIANCE & POLICY			SECURITY FEATURES & DESIGN			CODE REVIEW			SOFTWARE ENVIRONMENT		
[CP1.1]	12	19	[SFD1.1]	17	17	[CR1.2]	12	18	[SE1.1]	7	14
[CP1.2]	15	18	[SFD1.2]	13	15	[CR1.4]	13	18	[SE1.2]	16	18
[CP1.3]	9	15	[SFD2.1]	4	6	[CR1.5]	5	9	[SE1.3]	0	8
[CP2.1]	6	12	[SFD2.2]	6	12	[CR1.7]	3	13	[SE2.2]	5	5
[CP2.2]	5	8	[SFD3.1]	1	3	[CR2.6]	1	5	[SE2.4]	4	4
[CP2.3]	7	14	[SFD3.2]	3	8	[CR2.7]	5	5	[SE2.5]	1	7
[CP2.4]	5	11	[SFD3.3]	0	0	[CR2.8]	7	8	[SE2.7]	0	4
[CP2.5]	9	13				[CR3.2]	0	2	[SE3.2]	2	2
[CP3.1]	5	8				[CR3.3]	0	2	[SE3.3]	2	1
[CP3.2]	6	3				[CR3.4]	0	0	[SE3.6]	0	1
[CP3.3]	1	2				[CR3.5]	0	0	[SE3.8]	0	0
TRAINING			STANDARDS & REQUIREMENTS			SECURITY TESTING			CONFIG. MGMT. & VULN. MGMT.		
[T1.1]	10	14	[SR1.1]	13	17	[ST1.1]	17	16	[CMVM1.1]	17	18
[T1.7]	7	15	[SR1.2]	14	18	[ST1.3]	17	18	[CMVM1.2]	19	17
[T1.8]	3	12	[SR1.3]	13	18	[ST1.4]	7	12	[CMVM2.1]	16	17
[T2.5]	5	9	[SR2.2]	8	15	[ST2.4]	1	2	[CMVM2.2]	12	15
[T2.8]	3	6	[SR2.4]	5	14	[ST2.5]	0	3	[CMVM2.3]	9	13
[T2.9]	1	9	[SR2.5]	4	9	[ST2.6]	4	3	[CMVM3.1]	0	0
[T2.10]	0	3	[SR2.7]	3	8	[ST3.3]	1	2	[CMVM3.2]	1	1
[T2.11]	0	5	[SR3.2]	5	5	[ST3.4]	0	1	[CMVM3.3]	1	4
[T3.1]	0	2	[SR3.3]	3	4	[ST3.5]	2	1	[CMVM3.4]	1	6
[T3.2]	0	3	[SR3.4]	8	6	[ST3.6]	0	0	[CMVM3.5]	0	1
[T3.5]	0	4							[CMVM3.6]	0	0
[T3.6]	1	1							[CMVM3.7]	0	0
									[CMVM3.8]	0	0

FIGURE 28. BSIMM13 REASSESSMENTS SCORECARD ROUND 1 Vs. ROUND 3. This chart shows how 19 SSIs changed between their first and third assessments. Dark orange shows the top five activities with the most increase in observations by count. Light orange shows the next five activities with the most increase in observations by count.

G. DATA ANALYSIS: SATELLITE (SECURITY CHAMPIONS)

A security champions program is an organized effort to deputize members of the development community into being software security leaders for their geographies, application teams, or technology groups. Once they are inducted into the program, the SSI provides the champions with training, support, and the access needed to answer security questions.

- A security champions program allows an SSI and SSG to scale their reach throughout the organization and harmonize everyone's approach to software security.
- You can use this information to help justify your own outreach program.

A security champions program is an effective way to address the people and culture portions of the people, process, technology, and culture view of an SSI's scope. Firms typically rely on their security champions to lead the ground-level security push among developers, architects, QA, operations, and other stakeholders such as cloud and site reliability. A strong security champions program enables an SSI to scale people-driven activities, tune automated activities, and prioritize remediation tracking activities within an organization. In Figure 29, the green bars show that firms can achieve higher scores even with a lower ratio of SSG to developers (e.g., the bottom 20% have an SSG-to-developer ratio of 2.8%). One way these firms are able to scale is by increasing the ratio of champions to developers, as shown by the blue bars (e.g., the bottom 20% have a satellite-to-developer ratio of 1.4%).

While the presence of a champions program doesn't guarantee a high number of activity observations, there is a correlation that appears when grouping BSIMM firms by scores. More than 80% of firms in the highest scoring group have a champions program as compared to 20% in the lowest scoring group. Figure 30 shows the score increases from an average of 22.1 activities in the lowest scoring group (shown on the black line), up to an average of 70.5 activities in the high scoring group (shown here as the top 20%).

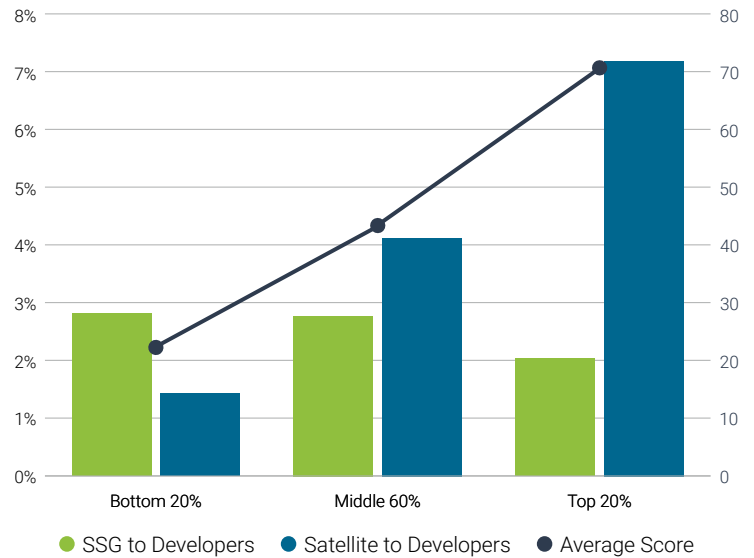


FIGURE 29. AVERAGE RATIO OF SSG AND SATELLITE SIZE TO DEVELOPERS FOR THREE SCORE BUCKETS. There is a strong correlation between security champions' support and overall BSIMM score (scale on the right). Note: For the group that consists of the middle 60% of firms, we left out a single outlier with a large SSG size that would have increased the ratio of SSG to developers from 2.8% to 3.4% for the entire group.

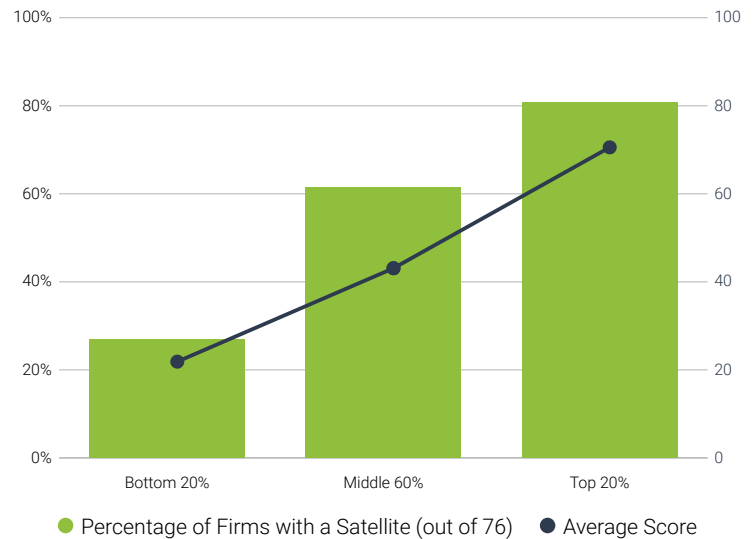


FIGURE 30. PERCENTAGE OF FIRMS THAT HAVE A SATELLITE, ORGANIZED IN THREE BUCKETS BY BSIMM SCORE. Presence of a satellite and average score (scale on the right) appear to be correlated, but we don't have enough data to say which is the cause and which is the effect.

When separating firms into groups with and without a satellite, the activity observation rate increases in every practice (see Figure 31). While the biggest differences between the two spiders are in Strategy & Metrics and Training, the firms with a satellite also spend consistently more effort on defect discovery in the Architecture Analysis, Code Review, Security Testing, and Penetrating Testing practices.

Figure 32 shows that as SSIs get older, they have higher average scores and are more likely to have a satellite (champions team). So is the presence of a satellite the reason for higher score or the consequence of older SSIs? One way to answer this question is to look at the average ratio of SSG size to number of developers, shown in Figure 29, which might indicate that there is a correlation between SSI reach and the size of the security champions team.

More than 80% of firms in the highest scoring group have a champions program, compared to 20% in the lowest scoring group.

Seventy-six percent of the 54 BSIMM13 firms that have been assessed more than once have a satellite, while 54% of the firms on their first assessment do not. Many firms that are new to software security take some time to identify and develop a satellite. This data suggests that as an SSI matures, its activities become distributed and institutionalized into the organizational structure, perhaps even into engineering automation as well, requiring an expanded satellite to provide expertise and be the local voice of the SSG.

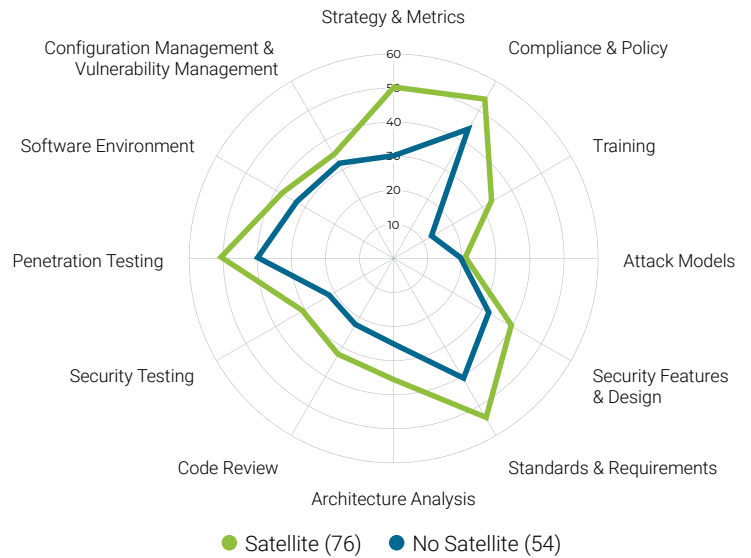


FIGURE 31. COMPARING FIRMS WITH AND WITHOUT A SATELLITE. The presence of a satellite (champions program) seems to correlate strongly with an increase in program maturity as evidenced by increased scores by practice on a percentage scale.

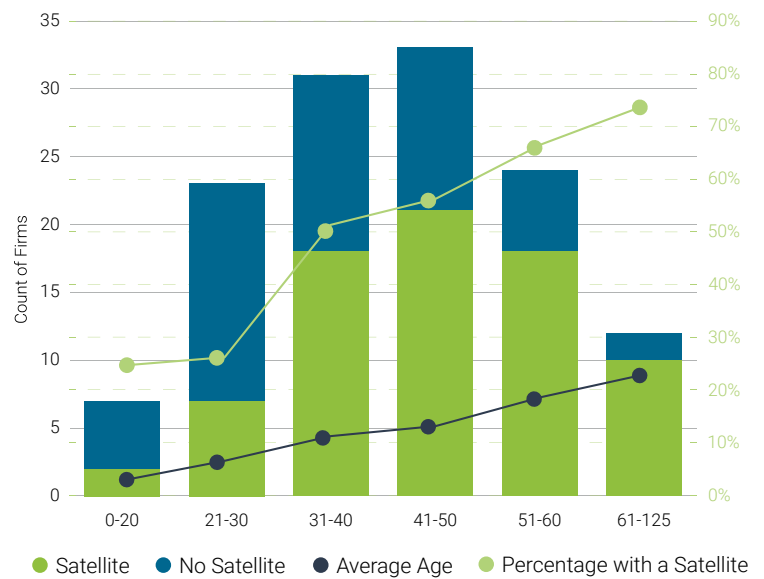


FIGURE 32. BSIMM SCORE DISTRIBUTION RELATIVE TO SATELLITE SIZE AND SSG AGE. Older SSIs (black line) not only tend to have a higher BSIMM score (buckets 0-20, 21-30, etc.), they are also more likely to have a champions program (green line).

H. DATA ANALYSIS: SSG

This section analyzes how SSIs evolve over time by analyzing SSG age, SSG score, and other relevant data.

SSGs are the primary implementers of an SSI, responsible for governance, enablement, productivity, and continuous growth. You can use this information to put your SSI and SSG on a growth path.

SSG CHARACTERISTICS

As the BSIMM community evolved, we added a greater number of firms with newer SSIs and began to track new verticals that have less software security experience (see Table 12 in Appendix E). Thus, we expected a decrease in participant scores, which is easily seen in Figure 33 for BSIMM6 through BSIMM8.

In BSIMM9, the average and the median scores started to increase. We saw the largest increase in BSIMM13 when the average and median scores increased by 4.1 and 3, respectively. One reason for this change in average data pool score appears to be the mix of firms using the BSIMM as part of their SSI journey. For example, Figure 34 shows how the SSG age of firms entering the BSIMM community changed over time. In BSIMM13, and in concert with the increase in average scores seen for BSIMM13 in Figure 33, we saw a significantly higher average and median SSG age of new firms versus what was seen in previous years.

A second reason appears to be firms continuing to use the BSIMM to guide their initiatives. Firms using the BSIMM as an ongoing measurement tool are likely also making sufficient improvements to justify the ongoing creation of SSI scorecards. See Appendix F for more details on how SSIs evolve as seen through remeasurement data.

A third reason appears to be the effect of firms aging out of the data pool (see Figure 35).

We see a similar assessment score trend in mature verticals such as that of the Financial vertical (see Figure 36).

Note that when creating BSIMM11, we recognized the need to realign the Financial vertical. Over the past several years, financial and FinTech firms differentiated significantly, and we became concerned that having both in one vertical bucket could affect our analysis and conclusions. Accordingly, we created a FinTech bucket and removed FinTech firms from the financial bucket. This action created a new FinTech vertical for analysis and reduced the size (but increased the homogeneity) of the Financial vertical. To be clear, we did not carry this change backward to previous BSIMM versions, meaning that some BSIMM10 and older financial data is not directly comparable to BSIMM11 and newer data.

Given their importance to overall SSI efforts, we also closely monitor satellite trends. Many firms with no satellite continue to exist in the community, which causes the median satellite size to be 9.5 (54 of 130 firms had no satellite at the time of their current assessment); 44% of the 27 firms added for BSIMM13 had no satellite at assessment time, as seen in Figure 37).

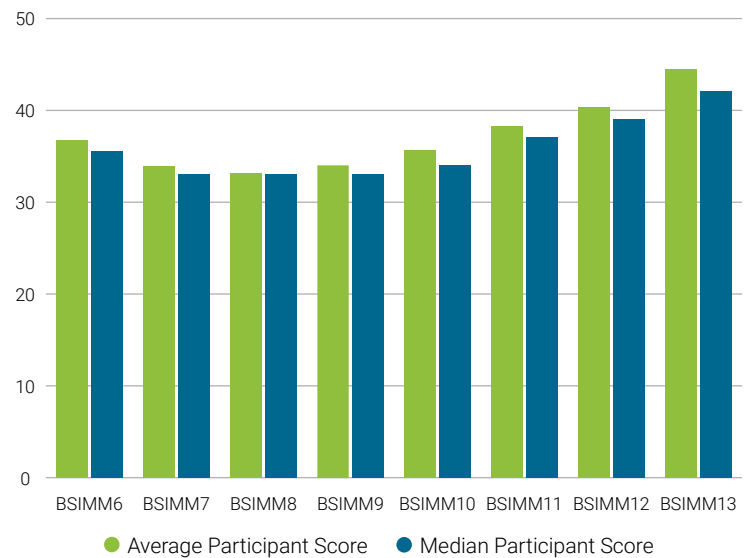


FIGURE 33. AVERAGE BSIMM PARTICIPANT SCORE. Adding firms with less experience decreased the average score from BSIMM6 through BSIMM8, even as remeasurements have shown that individual firm maturity increases over time.

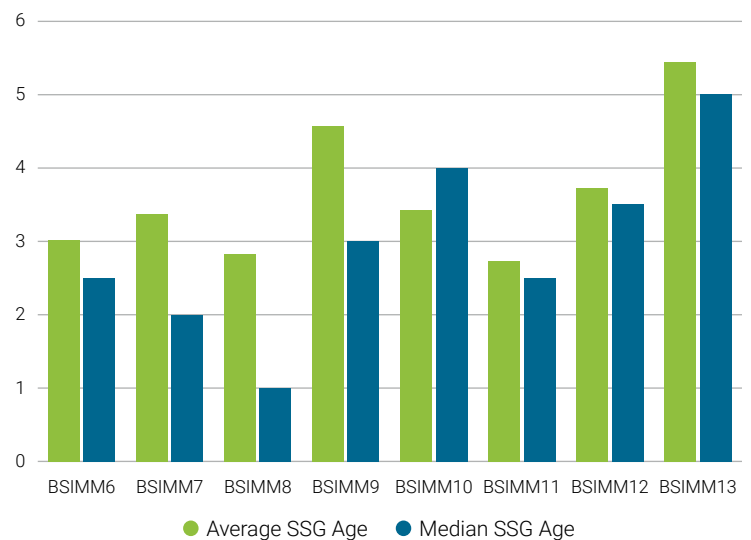


FIGURE 34. AVERAGE AND MEDIAN SSG AGE FOR NEW FIRMS ENTERING THE BSIMM DATA POOL. The median SSG age of firms entering BSIMM6 through BSIMM8 was declining and so did the average BSIMM score, while outliers in BSIMM7 and BSIMM8 resulted in a high average SSG age. Starting with BSIMM9, the median age of firms entering the BSIMM was higher again, which tracks with the increase of average BSIMM scores.

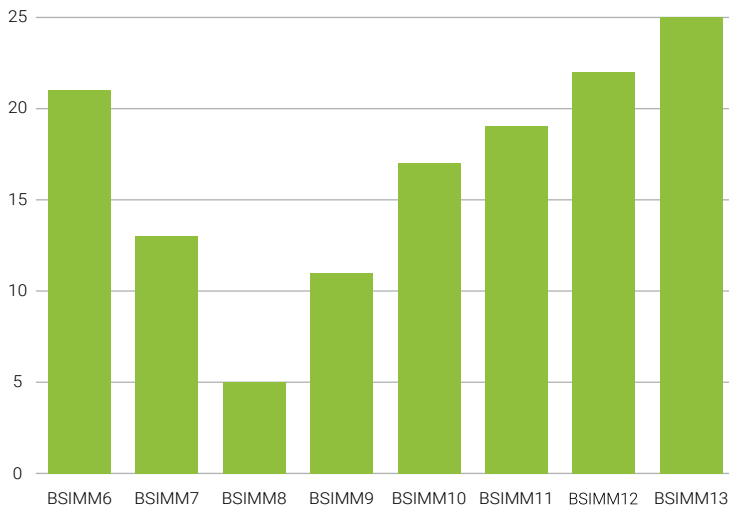


FIGURE 35. NUMBER OF FIRMS AGED OUT OF THE BSIMM DATA POOL. A total of 138 firms have aged out since BSIMM-V. Fourteen firms that had once aged out of the BSIMM data pool have subsequently rejoined with a new assessment.

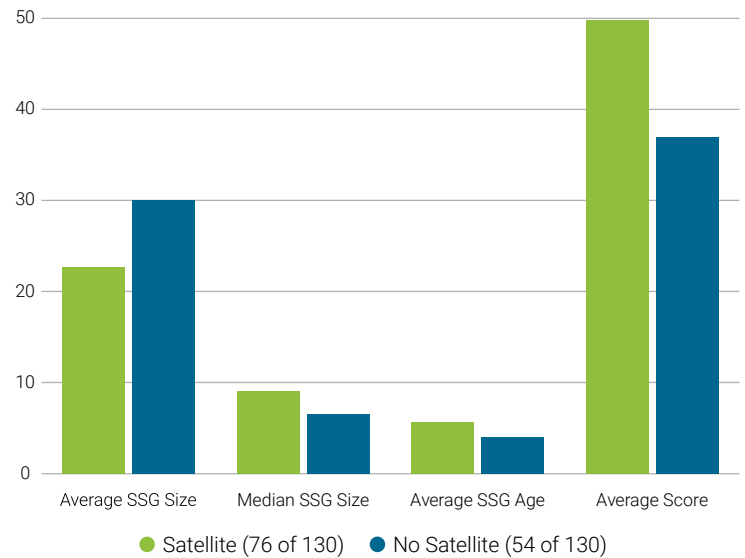


FIGURE 37. STATISTICS FOR FIRMS WITH AND WITHOUT A SATELLITE. This data appears to validate the notion that having more people, both centralized and distributed into engineering teams, helps SSIs achieve higher scores. For the 76 BSIMM13 firms with a satellite at last assessment time, the average satellite size was 112 with a median of 40 (not shown). We present the average and median SSG size to remove the impact of a few significant outliers.

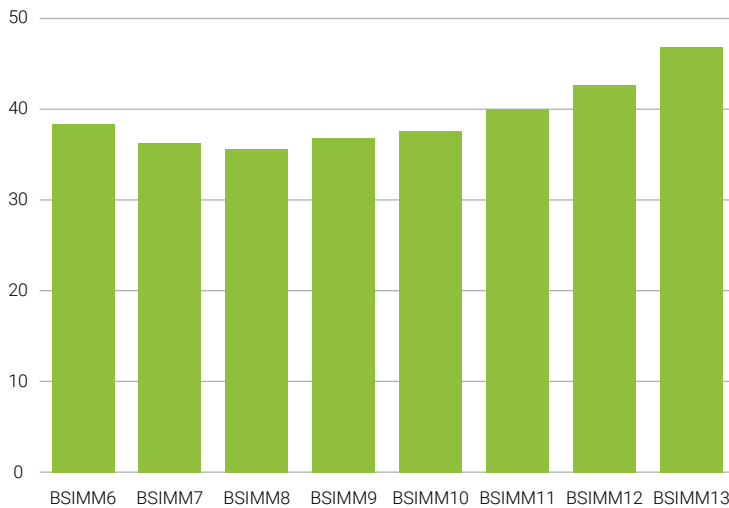


FIGURE 36. AVERAGE FINANCIAL VERTICAL FIRM SCORES. The average score across the Financial vertical followed the same pattern as the average score for AllFirms (shown in Figure 33). Even in such a mature vertical, we observe a rise in the average scores over time. We saw the largest increase in average score for financial firms in BSIMM13.

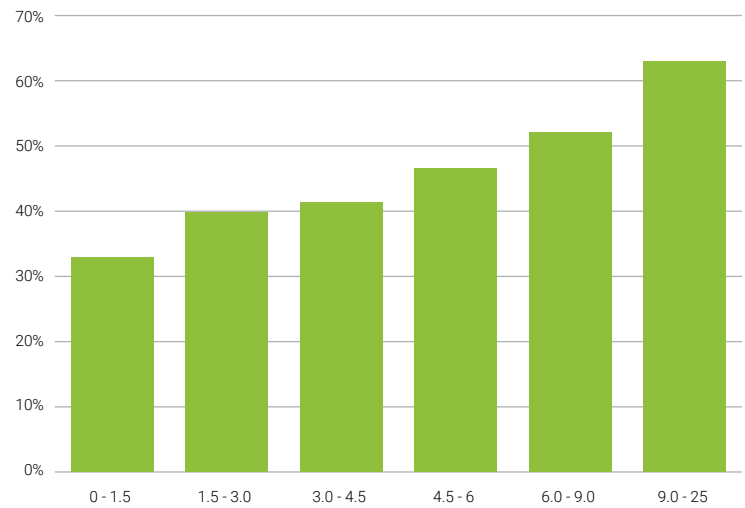


FIGURE 38. SSI SCORE DIVIDED BY AGE. By notionally organizing SSIs into emerging, maturing, and enabling phases by age in years, we see a steady growth in score as SSIs mature.

SSG CHANGES BASED ON AGE

This section analyzes how SSGs compare to each other based on their age. We've mentioned a trend that older SSIs generally achieve higher scores, and we show this trend in Figure 16 in Appendix D. Here, we analyze the data in more detail to identify additional trends related to SSG age.

For this analysis, we put the 130 BSIMM13 SSIs into six groups based on SSG age. Figure 38 shows the trend discussed earlier: the older the SSI, the higher its BSIMM score. While the journey through emerging, maturing, and enabling phases is not a straight line (see Appendix B), here we equate the emerging phase with the first two bars from the left (0-1.5 and 1.5-3.0 years of age), maturing phase with the next two bars, and enabling phase with the last two.

While Figure 38 provides a low-resolution view into how SSIs change with SSG age, the following five figures increase the resolution and compare the normalized spiders for SSIs organized by their age. Figure 39 shows, on a percentage scale, how the SSI is changing through its emerging phase. The green line shows what the program looks like when SSIs are initially organizing themselves and discovering what activities are already happening in the organization. At this point in the journey, we typically see a relatively high effort in Compliance & Policy, Standards & Requirements, and Penetration Testing. Likely, these efforts are already in place due to compliance obligations, an existing cybersecurity program and its focus on standards, and quick wins in defect discovery by leveraging penetration testing.

Over the next 18 months (blue line), SSIs build some capability around documenting and socializing the SSDL, publishing and promoting the process, and defect discovery for high-priority applications. The differences between two spiders in Strategy & Metrics, Compliance & Policy, Standards & Requirements, and Architecture Analysis result from these efforts.

As SSIs move toward the maturing phase, they start focusing on improving the efficiency, effectiveness, and scale of existing efforts, see the "Maturing an SSI: Harmonizing Objectives" section of Appendix B. This push typically involves getting more value out of existing activities rather than doing more activities. Figure 40 shows the difference in normalized spiders for organizations toward the end of their emerging phase (green line) and the beginning of their maturing phase (blue line).

The lack of any large differences between the spiders in Figure 40 shows that firms at this stage focus on tweaking the existing program as they improve scale, efficiency, and effectiveness. The changes are often an investment in quick wins (such as penetrating testing) and automation (such as code reviews). As shown in the diagram, when these SSIs look to improve scale and efficiency, they appear to have less time for manual efforts in the Architecture Analysis practice.

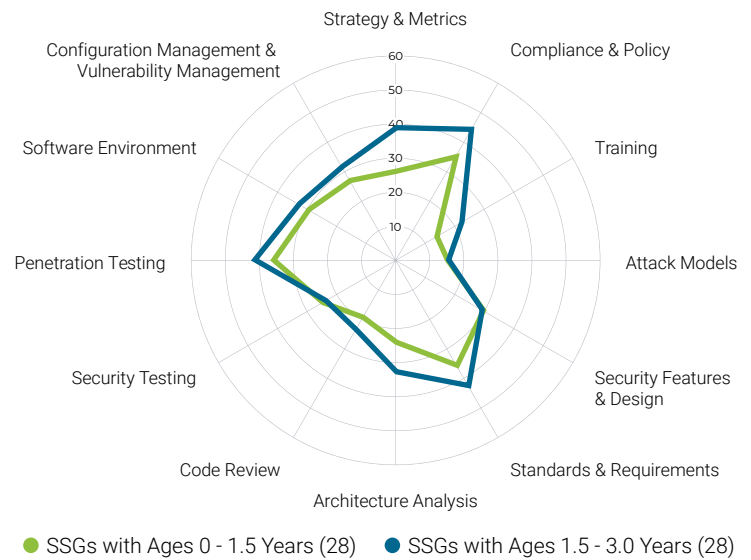


FIGURE 39. COMPARING EMERGING SSIs. As emerging SSIs move from initial discovery steps (green area) toward defining and rolling out the program (blue area), they invest in Strategy & Metrics, Compliance & Policy, Standards & Requirements, and Architecture Analysis. This tracks with recommendations in Appendix B on how to start an SSI, where almost 45% of all recommended activities in Figure 10 are from these four practices.

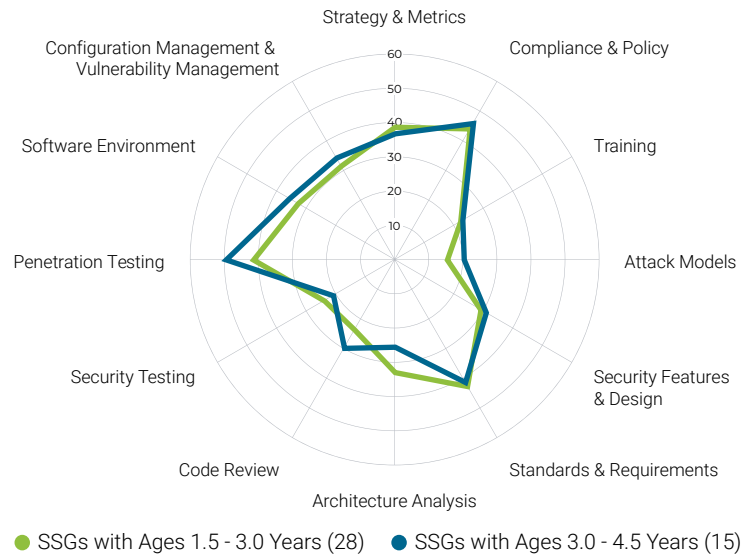


FIGURE 40. COMPARING LATE EMERGING TO EARLY MATURING SSIs. As firms move from emerging to maturing, the average score increase is relatively small. This aligns with our qualitative observations in Appendix B that these firms often focus more on the scale, efficiency, and effectiveness of existing activities in their SSIs versus working on implementing new activities.

As SSIs move toward the end of their maturing phase, they start investing again in improving policies, standards, requirements, processes, metrics, and evangelism as shown by significant differences in the spiders in Figure 41. The increase in observation rates in the Strategy & Metrics, Compliance & Policy, and Standards & Requirements practices demonstrate this trend.

We acknowledge that the number of firms in an enabling phase (i.e., in the higher age ranges) is smaller compared to the other age ranges. As such, some factors specific to verticals might significantly influence the overall shape of the spiders. For example, 43% of firms with an SSG age between six and nine years are in the Financial vertical as compared to 34% in the entire BSIMM13 data pool. Similarly, 39% of the firms with an SSG age above nine years are in the Technology vertical versus 25% in the entire data pool. As we analyze the next two figures, we keep these facts in mind. Refer to Appendix E for more analysis of how the verticals compare to each other.

One potential explanation for the dip in Security Testing shown in Figure 42 is that the Financial vertical has one of the lowest observation rates for this practice. For the spike in the Penetration Testing practice, almost 60% of all firms in the age bucket between six and nine years are either in Cloud, ISV, or FinTech verticals—the three verticals with the highest observation rates in the Penetration Testing practice. Outside of the outliers mentioned above, SSIs gradually increase their effort in all other practices as they start their enabling journey.

In Figure 43, we see some of the largest increases in observation rates, specifically in Strategy & Metrics, Attack Models, Security Features & Design, and Security Testing. The spike in Security Testing can be explained by the high percentage of technology firms in this age bucket. The average observation rate in the Security Testing practice is almost 2.5 times higher for technology firms compared to all other firms.

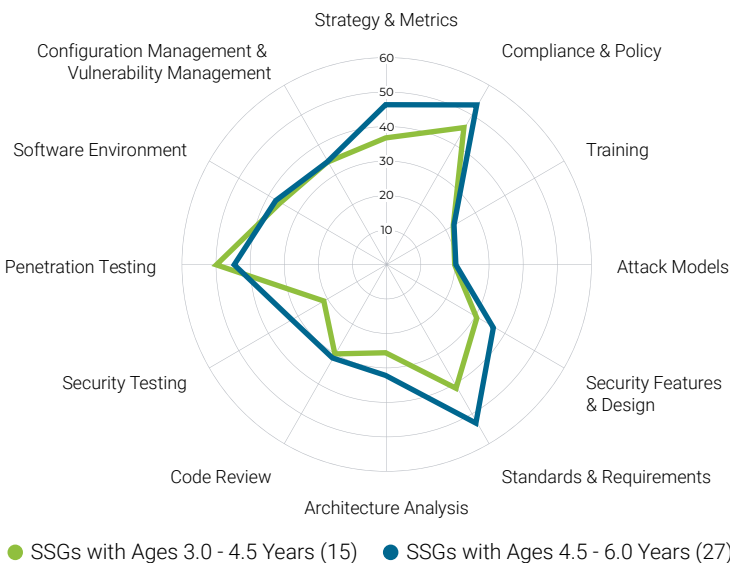


Figure 41. COMPARING MATURING SSIs. As firms move toward the end of their maturing journey, SSGs start focusing again on implementing new activities. Here, we see a trend toward a “shift left” approach where there is increased investment in the Architecture Analysis and Security Testing practices and decreased investment in the Penetrating Testing practice.

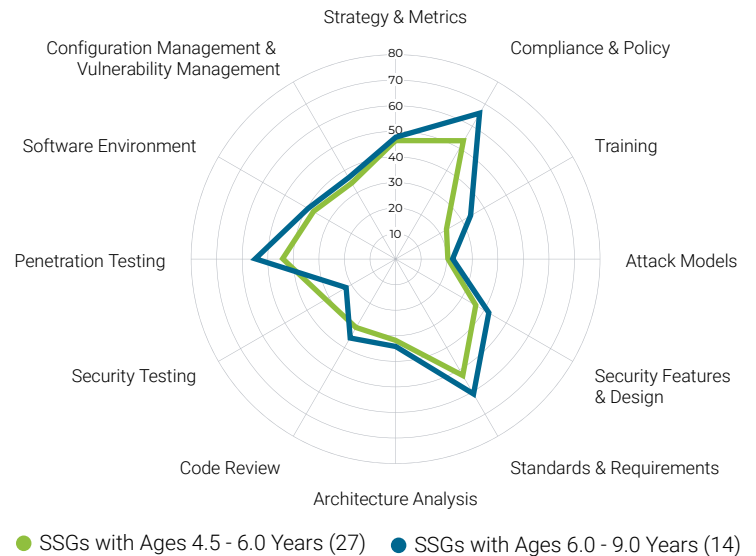


Figure 42. COMPARING LATE MATURING TO EARLY ENABLING SSIs. As firms move from the maturing to the enabling stage, SSIs continue to invest in Compliance & Policy. This stage is the first time that we see a significant investment in the Training practice. Overall, this comparative growth aligns with concepts such as putting “Sec” in DevOps as well as scaling outreach and expertise, which are discussed in the “Enabling SSIs” section of Appendix B.

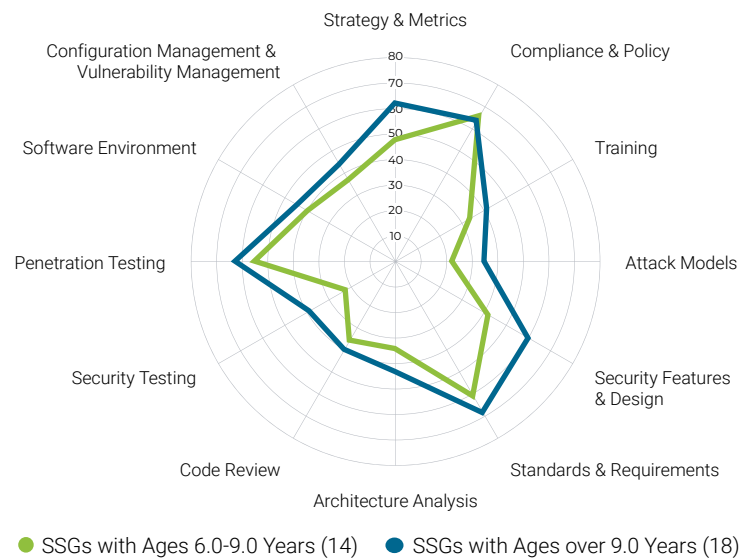


Figure 43. COMPARING ENABLING SSIs. As SSIs continue their enabling phase, they invest significant effort in reusable and pre-baked security controls (e.g., from the Security Features & Design practice) and learning from the attacker’s perspective (e.g., from the Attack Models practice). In fact, the increase in observation rate of activities in Security Features & Design is the highest increase in observation rates among all practices across all age buckets. This is also the first time we see significant increase in observation rate in the Attack Models practice.

BSIMM



13