BSIMM14

REPORT 2023

TABLE OF CONTENTS

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 to help others.

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 SSIs to build security into 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.

Since BSIMM1 in 2009, we've been early reporters on security program changes across people, process, technology, culture, compliance, digital transformation, and much more. Welcome to the BSIMM14 report, and thank you for reading.

WELCOME TO BSIMM14

If you're in charge of an SSI, understanding the BSIMM and its use by participants will help you plan strategic improvements. If you're running the 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 many organizations refer to as their application 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 inform their SSI improvements. Here, we present BSIMM14 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 more than 100 companies to your own will directly inform your strategy for improvement and growth.

BSIMM core knowledge is the activities we have directly observed in our participants—the group of firms that use the BSIMM as part of their SSI management. Each participant has their own unique SSI with an emphasis on the building 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 Part 5. The SSF comprises four domains—Governance, Intelligence, SSDL Touchpoints, and Deployment—with those domains currently composed of 126 activities. The Governance domain, for example, includes activities that fall under the organization, management, and measurement efforts of an SSI.

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

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.

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.** A group of interested and engaged developers, cloud security engineers, deployment engineers, architects, software managers, testers, or people in similar roles who have an active interest in software security and contribute to the security posture of the organization and its software.
- **Data pool.** The collection of assessment data from the current participants.
- **Domain.** One of the four categories the framework is divided into, i.e., Governance, Intelligence, SSDL Touchpoints, and Deployment.
- **Participants.** The group of firms in the current data pool.
- **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.

BSIMM14 DATA HIGHLIGHTS

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

Note: Items in italic green refer to specific BSIMM activities in Part 6.

Activities are the building blocks of the BSIMM, the smallest units of granularity 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 BSIMM14, we've made the following changes to the model based on what we see in the BSIMM data pool:

- We moved the activities *provide expertise via open collaboration channels, have a research group that develops new attack methods, monitor automated asset creation, identify open source,* and *track software defects found in operations through the fix process* because we now see them more frequently.
- We moved the activities *create technology-specific attack patterns* and *maintain and use a top N possible attacks list* because they're not growing as fast as other common activities in their practice area.
- We added the activity *protect integrity of development toolchains* because we are beginning to see this more.

Unique in the software security industry, the BSIMM project has grown from nine participating companies in 2008 to 130 in 2023, now with approximately 3,600 software security group (SSG) members and 7,500 security champions. The average age of the participants' SSIs is 5.2 years. The BSIMM project shows consistent growth even as participants enter and leave over time—we added 23 firms for BSIMM14 and dropped 23 others whose data hadn't been refreshed.

This 2023 edition of the BSIMM report—BSIMM14—examines anonymized data from the software security activities of 130 organizations across various verticals, including cloud, financial services, financial technology (FinTech), healthcare, independent software vendors (ISVs), insurance, Internet of Things (IoT), and technology organizations.

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 BSIMM14 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 *make code review mandatory for all projects* and *identify open source*, are mentioned in the Trends and Insights section. In addition, the *streamline incoming responsible vulnerability disclosure* activity introduced in BSIMM12 has the largest increase in observation count. Note that for some of the activities in Table 2, the growth in observation is a relatively new change. For example, the activity *have a research group that develops new attack methods* saw virtually no growth from BSIMM9-BSIMM12 but had a significant jump in observation rates in BSIMM13, and BSIMM14 has continued that climb. Two years of growth suggests the change is meaningful and the activities are worth considering for your program.

In BSIMM13, we reported new growth after little change over time in the *enforce security checkpoints and track exceptions* activity. This activity has continued to grow in BSIMM14 as firms are able to take advantage of modern automation options in the development pipeline.

In the other direction, in BSIMM13, we reported that the *have SSG lead design review efforts* activity saw continued growth for years but then decreased significantly for BSIMM13. In BSIMM14, this decrease has corrected, with a small growth in observations this year.

BSIMM14 TOP 10 ACTIVITIES	
PERCENT	**DESCRIPTION**
90.8%	Implement security checkpoints and associated governance.
90.0%	Create or interface with incident response.
87.7%	Identify privacy obligations.
87.7%	Use external penetration testers to find problems.
86.9%	Ensure host and network security basics are in place.
86.2%	Use automated code review tools.
84.6%	Perform edge/boundary value condition testing during QA.
83.1%	Perform security feature review.
79.2%	Unify regulatory pressures.
79.2%	Create a security portal.

TABLE 1. TOP ACTIVITIES BY OBSERVATION PERCENTAGE. The most frequently observed activities in BSIMM14 are likely important to all SSIs.

BSIMM14 TOP 10 ACTIVITIES GROWTH BY COUNT	
INCREASE	**DESCRIPTION**
15	Streamline incoming responsible vulnerability disclosure.
13	Implement cloud security controls.
12	Make code review mandatory for all projects.
11	Have a research group that develops new attack methods.
11	Define secure deployment parameters and configurations.
11	Use application containers to support security goals.
10	Schedule periodic penetration tests for application coverage.
9	Identify open source.
8	Document a software compliance story.
8	Enforce security checkpoints and track exceptions.

TABLE 2. TOP ACTIVITIES BY RECENT GROWTH IN OBSERVATION COUNT. These activities had the largest growth in BSIMM14, out of 32 firms measured during the last 12 months, which means they are likely important to your program now or will be soon.

TRENDS AND INSIGHTS SUMMARY

These BSIMM trends and insights are a distillation of software security lessons learned across 130 organizations that collectively have 11,100 security professionals helping about 270,000 developers do good security work on about 97,000 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 of firms taking advantage of security automation over manual subject-matter expert (SME)-driven security activities. Over time, we've seen a trend in testing being applied throughout the software lifecycle ("shift everywhere"), 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: Trends and Insights later in this document for more.

How Software Security Is Changing

Organizations are modernizing development toolchains to give their developers the best tools for building software. Security leaders are taking advantage of the easy-to-use yet powerful automation available in these toolchains to update security testing and touchpoints. This is allowing shift everywhere as a philosophy to move beyond testing to decisions and governance.

When automation makes security tasks easier, trends emerge around automated activities. Modern toolchains, for example, allow for security testing in the QA stage to be automated, much like SAST scans that happen earlier in the development process. This has led to a 10% growth in the *integrate opaque-box security tools into the QA process* and *include security tests in QA automation* activities.

Security teams that embraced the shift everywhere testing philosophy found that their pipelines were able to take scripted actions based on the results of those automated security tests. The automated decisions enabled by these pipelines led to a 60% growth in the *integrate software-defined lifecycle governance* activity in the past year.

Firms are also using automation to better gather and make use of intelligence provided by sensors in the pipeline. Observations of firms that *build a capability to combine AST results* have nearly doubled. Additionally, the use of captured knowledge by the *enforce secure coding standards* activity is again seeing growth after a period of decline.

Finally, some firms are using the insights gleaned from sensors throughout the development lifecycle to proactively prevent vulnerabilities before they become an issue for developers. *Drive feedback from software lifecycle data back to policy* was observed at an increased rate of 36% in the past year, further assisting the engineers who drive the development lifecycle.

What Is Shift Everywhere, Really?

To define shift everywhere, let's start by stating what it's not: it's not trying to do all the security things everywhere in all parts of the software lifecycle (SLC) all the time. Instead, shift everywhere is a philosophy; it's an approach to SLC governance that acknowledges the reality that consistently achieving acceptably secure software is a shared responsibility, and that this responsibility traverses legal, audit, risk, governance, IT, cloud, technology, vendor management, and resilience, among others. Each stakeholder has their own business processes to execute, but each also needs to do their version of security sign-off, which requires understandable and usable telemetry from the SLC toolchain.

Not so very long ago, the only view into the SLC afforded to stakeholders was, "We shipped it yesterday!" That was horrible then and is much worse now, mainly because automation generates telemetry that is easy to route to stakeholders. A shift everywhere approach starts by asking how these roles get the information they need, when they need it, in the processes they normally use, with little or no additional friction, then it bridges that gap, giving each role access to appropriate sensors, whenever they need it, from anywhere in the SLC. Shift everywhere is a philosophy about the security testing and sensors that generate information for all stakeholders in the company, it's not rooted in increasing the security spend or effort. Accordingly, shift everywhere is not adding more security for security's sake, instead, it's ensuring that every security stakeholder can knowledgably make risk management decisions.

Expanding Security's Scope

External pressures like government regulations and increased awareness of supply chain threats are leading organizations to extend risk management to the software that they integrate from outside sources, the toolchains used by their developers, and the software present in their operating environments. We have added the new activity *protect integrity of development toolchains* to begin tracking how firms protect software and artifacts as they pass through their development pipeline.

The first step many firms take in understanding the risk they're bringing into their software by integrating third-party and open source components is scanning with a software composition analysis tool. These moment-in-time checks allow security teams to uncover newly published vulnerabilities in software. After scanning all of the integrated components, teams *create bills of materials for deployed software*, observations of which grew by 22% from BSIMM13 to BSIMM14.

After scanning individual projects and compiling software bills of materials (SBOMs), firms seek to take a more holistic approach to managing open source risk across the portfolio. Two activities associated with this portfolio-wide risk management, *identify open source* and *control open source risk*, both saw just under 10% growth from BSIMM13 to BSIMM14.

Firms are also getting tough on vendors and expecting the software they buy to be secure at the time of acceptance. Observations of the *ensure compatible vendor policies* activity, which reflects how firms enforce security standards on organizations that provide bought and bespoke software, grew by 21% as firms held vendors to similar standards as they use internally.

Who Owns Security

In a trend a decade in the making, we see a growing number of organizations referring to their centralized effort as a product security program (vs. application or software security). We measure this by noting where SSI reporting chains pass through a Chief Product Officer, VP of Products, or Product Security Manager, which now accounts for almost a quarter of the data pool (31 of 130 firms). This naming trend seems to correlate with product vendors creating security programs to manage the risk associated with software that leaves the organization to exist in hostile environments for years to decades (as compared to applications in private data centers).

Initially, product security teams were formed to deal with the unique attack surfaces of their products compared to the web applications in heavy use in financial verticals. Firms continue to deal with unique threats with *create technology-specific attack patterns*, an activity that has grown by 15% since BSIMM13.

Understanding and building technology-specific guidance in the absence of industry best practices for products with unique operating requirements is the first step in securing software that exists in uncontrolled or potentially dangerous locations. To deal with vulnerabilities discovered after software is deployed to external environments, security teams will stand up a Product Security Incident Response Team (PSIRT) function to handle communication about and reactions to reported vulnerabilities. Observations show that the associated *streamline incoming responsible vulnerability disclosure* activity is now present in more than a quarter of the BSIMM14 data pool.

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

Important Decisions in Software Security

For such a complicated endeavor, software development and its associated security governance is simple on paper: write some code, then build it, applying all the security testing there was time for. Development fixed the worst security defects discovered, with some of the remainder becoming requirements for the next release. However, actually performing all those steps in the real world can be expensive in terms of hours spent on manual processes. BSIMM data shows some of the decisions made by firms that can help scale security in spite of those expenses.

The oldest insight provided by BSIMM data is that the decision to build and operate a security champions program has a measurable impact on total BSIMM scores. In BSIMM14, firms with security champions scored on average 25% higher than firms without one. Observations of training activities such as *conduct software security awareness training*, *deliver on-demand individual training*, and *include security resources in onboarding* were also positively correlated with the presence of a security champions program.

Joining security champions as an enabler of security capabilities is the organizational decision to target cloud architectures. When we assess firms that *implement cloud security controls*, we also see scoring gains in the Compliance & Policy and Software Environment practices of 21% and 44%, respectively.

While cloud architectures have made certain security activities easier and more affordable for firms, recent economic conditions have caused a reduction in expensive, SME-driven activities that are not easy to automate. Observations of *build attack patterns and abuse cases tied to potential attackers* declined by 25%, *use centralized defect reporting to close the knowledge loop* shrank 18%, and *maintain and use a top N possible attacks list* decreased by 31%.

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 participants, with various parts likely right for your program as well.

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 a 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 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 SBOMs, API and microservices lists, various as-code artifacts, 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 might require 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.

Make the Right Investments

- 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, AI, 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.

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 is 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 BSIMM14 data, review Appendix D.
- If you want to look at our analysis of the BSIMM data, review Appendices E though H.

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 BSIMM14. A list of software security controls can be a very helpful guide here; the BSIMM project has worked since 2008 to ensure that its content matches real-world efforts.

GOVERNANCE		
STRATEGY & METRICS	COMPLIANCE & POLICY	TRAINING
• 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.	• 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.	• 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. • Provide expertise via open collaboration channels. • Reward progression through curriculum. • Provide training for vendors and outsourced workers. • Identify new satellite members (security champions) through observation.

INTELLIGENCE		
ATTACK MODELS	SECURITY FEATURES & DESIGN	STANDARDS & REQUIREMENTS
• 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. • Collect and publish attack stories. • Build an internal forum to discuss attacks. • Have a research group that develops new attack methods. • Monitor automated asset creation. • Create and use automation to mimic attackers. • Create technology-specific attack patterns. • Maintain and use a top N possible attacks list.	• 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 to approve and maintain secure design patterns. • Require use of approved security features and frameworks. • Find and publish secure design patterns from the organization.	• Create security standards. • Create a security portal. • Translate compliance constraints to requirements. • Identify open source. • Create a standards review process. • 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**
• 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 design patterns. • Make the SSG available as an AA resource or mentor.	• 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.	• 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**
• 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.	• 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. • Protect integrity of development toolchains.	• Create or interface with incident response. • Identify software defects found in operations monitoring and feed them back to engineering. • Track software defects found in operations through the fix process. • Have emergency response. • Develop an operations software inventory. • Fix all occurrences of software defects found in operations. • Enhance the SSDL to prevent software defects 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 126 activities are organized into the 12 BSIMM practices, which are within four domains.

PART 2:
TRENDS AND INSIGHTS

TRENDS AND INSIGHTS

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 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 in your vertical are doing to then inform your own strategy.

The past year has ushered in many changes for the software security industry. Artificial intelligence (AI) and large language models (LLMs) have burst onto the scene, and in addition to being integrated into products, they're now used to design applications and hardware, to create and test software, and in all other parts of the software lifecycle. Governments the world over are demanding (yet again) that companies create software security programs, account for and secure the software that's integrated into their products, and continually address software supply chain risk. Companies behind DevSecOps platforms, cloud solutions, and security tooling are rising to the challenges from both the marketplace and attackers to make it easier for developers to automate security tooling and processes. All of this is happening under economic conditions that see shrinking software security budgets that make it difficult for firms to maintain their level of security while cutting expensive SME-driven activities.

As part of their mitigation tactics, many organizations are maturing their automation to go beyond defect discovery, expanding their scope to minimize the risk introduced by supply chains, taking a holistic approach to securing their applications and products, and leveraging capabilities that make security possible under these evolving conditions. They're also increasingly adding AI into their ecosystems, which can increase productivity but also introduces new attack surfaces and risk. We're continuing to watch these and other developments.

Evolution of Shift Everywhere

Twenty years ago, organizations took notice of the excessive costs, efforts, and risk associated with testing for security defects only just before promotion to production. This large friction point helped drive the development and adoption of SAST tooling, which drove the software industry to shift testing to the left in the SDLC, a place where vulnerabilities could be found and fixed faster and for less money. More than 10 years ago, shift left was expanded to a broader testing philosophy where firms would also test designs and other development artifacts (e.g., golden masters, configurations, anything done as-code) as soon as they were ready—this was the beginning of shift everywhere. As firms moved some of their security efforts into engineering toolchains and processes, thereby empowering developers with the best tooling available to enable DevSecOps transformations, they also adapted shift everywhere testing methods into their automated and mature tooling. Security teams began automating their workflows as soon as developers adopted modern platforms, allowing defect discovery to transform from a manual process to something more set-and-forget. As platforms matured, firms began to not only check for internal and external governance compliance in the pipeline but also began enforcing security decisions automatically (e.g., security sign-off of coding standards adherence). Today, firms that have embraced the culture of shift everywhere in the pipeline are updating policy and strategy to integrate security touchpoints as-code throughout the SDLC.

Integrating Tooling

Firms are integrating tooling that enables developers to take a more active role in QA testing cycles, and security-minded developers are expanding security activities in this direction as well. Observations of the *integrate opaque-box security tools into the QA process* and *include security tests in QA automation* activities both rose by about 10% from BSIMM13 to BSIMM14. While already one of the top activities, *use automated code review tools* also rose by about 5% in the same period as developers took advantage of the automated SAST tooling included in modern CI/CD pipeline solutions. This security automation is the bedrock that the virtuous cycle of modern shift everywhere runs on.

Governance and Automation

As shift everywhere matured, firms began to automate pipeline security decisions in response to security findings from automated tooling. The activity *integrate software-defined lifecycle governance* was introduced in BSIMM10 and has seen slow but steady progress, growing 60% in the past year. Additionally, because it is bad practice to mandate unsustainable security measures, the ease of automation has removed that barrier and allowed firms to *make code review mandatory for all projects*, with an observation increase of about 68% since BSIMM10. Automating decisions and governance allows firms to manage risk in real time.

Security Touchpoints

Achieving well-secured software is more than just finding defects and breaking the build, given the variety of opportunities for security touchpoints to enhance security in the SSDL. Automation is enabling firms to implement the shift everywhere philosophy of right-sized testing at the right time in native processes, and we see that the *implement event-driven security testing in automation* activity has grown from 2 to 6 observations in the past two years. Firms are also getting smarter via activities like *build a capability to combine AST results*, which has nearly doubled, and *enforce secure coding standards*, which is again seeing growth after a period of decline. Shifting security touchpoints everywhere via automation represents the next phase of the shift everywhere philosophy.

Enabling People

The benefit of embracing shift everywhere is that it frees up people to do what they do best: being creative, solving problems, and building things. Firms that have taken full advantage of automation can then *drive feedback from software lifecycle data back to policy* and have done so at an increase of 36% in the past year to better enable the developers who drive the SDLC. Another way to make decisions that relieves the security demands placed on developers is to eliminate classes of vulnerabilities proactively, with observations of the *fix all occurrences of software defects found in operations* and *enhance the SSDL to prevent software defects found in operations* activities growing by just over 25%. By using automation as part of their DevSecOps culture, firms are leveraging toolchains to make security easier and creating more free time for humans.

SOFTWARE SUPPLY CHAIN RISK MANAGEMENT

Last year, we reported on how Executive Order 14028 and supply chain attacks led to an increased focus on managing risk in the software supply chain. To reflect observed efforts to not be the weak link in the software supply chain, we added a new activity, *protect integrity of development toolchains*, to begin tracking how firms protect software and artifacts as they pass through their development pipeline. Over the past year, more firms have expanded their security programs to formally address supply chain risks by accounting for risk in the software they purchase from vendors and procure from open source projects.

Software Bill of Materials (SBOM)

Firms often take the first step in understanding which application components come through the supply chain by creating a bill of materials of included libraries and dependencies for the software they're using or integrating. Organizations are slowly building SBOMs, with a 22% increase in observations of the *create bills of materials for deployed software* activity from BSIMM13 to BSIMM14. In BSIMM13, we introduced *the perform application composition analysis on code repositories* activity and have two observations so far. While SBOMs were initially found to be useful when responding to critical vulnerabilities in open source libraries, organizations are also using them to make informed risk decisions about what they are including in their production software.

Open Source Risk Management

Organizations have been incorporating open source projects into their software for decades, but OSS risk management has been a widespread priority for only the past five years or so. Activities associated with OSS risk management, *identify open source* and *control open source risk*, both saw just under 10% growth from BSIMM13 to BSIMM14. Identifying and controlling open source in use by developers is vital to safely taking advantage of the many benefits of OSS use.

Vendor Management and Bespoke Software

BSIMM14 data shows that firms are changing their relationships with vendors and expecting vendors to be more mature in how they build secure software. While observations of the *ensure compatible vendor policies* activity, which reflects how firms enforce security standards on organizations that provide bought and bespoke software, grew by 21% in the past year, activities around *create SLA boilerplate* and *include software security SLAs in all vendor contracts* both saw no growth. In addition, there were small declines in *provide training to vendors and outsourced workers* and in *communicate standards to vendors*. More firms are expecting vendors to supply software that meets or exceeds their security expectations without the additional effort of security coaching, training, or hand-holding.

PRODUCT SECURITY AND APPLICATION SECURITY

Commercial product firms historically have had unique security requirements not faced by other verticals in that their software must exist outside of safe, protected, and controlled data centers. This, in turn, drives differences in their required post-deployment security capabilities. After steady growth for over a decade, over one-quarter of firms (35 of 130) now have SSI reporting chains that pass through a Chief Product Officer, VP of Products, or a Product Security Manager. The change in nomenclature from application and software security titles appears to originate in a desire to focus their SSIs on specific capabilities tied to their unique software lifecycle.

Product-driven Security Requirements

Some of the activities with the largest amount of growth in the BSIMM14 data pool aid in providing security guidance to developers who produce software with risk profiles different from traditional web applications. To secure software with unique risk profiles, firms have worked toward understanding which attack surfaces and methods exist and then building new standardized design patterns that are resilient against those attacks. Observations of *have a research group that develops new attack methods* and *drive analysis results into standard design patterns* have both doubled since BSIMM13. Additionally, the *identify potential attackers* and *create technology-specific attack patterns* activities have each grown by just over 15%. Understanding the attacks that products will be subject to is the first step in building software that will be resilient to the risk inherent in the unique environments where those products will be installed.

Shipping Products to Dangerous Environments

While firms in certain verticals typically deploy developed software to data centers or cloud environments that they control, many product companies have little or no ongoing control of their deployed software. This scenario sets different requirements for post-deployment risk management. As the representation of product security programs in the BSIMM data pool has grown, the *streamline incoming responsible vulnerability disclosure* activity, which is a major function of the PSIRT capability, has grown to more than 25% of the data pool in just two years. Additionally, developers can prepare their products for client-controlled environments, as seen in observations of the *protect code integrity* and *define secure deployment parameters and configurations* activities, each of which grew by nearly 30% in the same period.

Growing "Product Security Program" Representation

According to historical BSIMM data, the first product security program (as judged by reporting chain titles) didn't show up until BSIMM4. Since then, the representation of SSIs led by someone with a product security title, or who reports through a VP or Director of Product Security or a Chief Product Officer, has grown from 2% in BSIMM4 to 26% in BSIMM14. From BSIMM6 through BSIMM10, the representation of these product security leadership roles was relatively flat at 9-10% of the data pool. In BSIMM11, that number jumped to almost 15% and has grown to 26% today. There are many reasons why commercial product firms are focusing on software security, ranging from external drivers like regulations from the FDA or other government agencies, internal pressures to enhance product feature sets by embedding connected software, and pressure from customers and the United States government.

SECURITY ENABLERS

As organizations seek to modernize their software security programs, there are actions they can take that happen outside of the SSG that have a positive impact on the larger application security posture. Historically, the trend that has stayed true the longest is that firms with security champions (or satellite) programs are able to integrate a greater number of BSIMM activities. However, this trend is now joined by a second security enabler in the adoption of cloud architecture. Companies have moved to cloud environments to gain cost savings, dynamically scale capacity, and take advantage of modern features without costly data center upgrades. Security teams have also seen gains as their firms move to the cloud.

Security Champions

Security champions programs have long been an enabler for software security teams. A security champion is usually a developer, QA tester, or architect who is deputized into an enabler role and provided with additional training and security resources to be the local security professional in a development team. BSIMM14 firms with a security champions program (80 of 130 firms) score on average 25% higher (13 observed activities) than firms without one (50 of 130). This aspect of shared responsibility is crucial to scaling distributed security tasks such as tool automation, security defect triage and remediation, and incident response. Additionally, in BSIMM14, programs with security champions had several training activities that were present at a much higher rate than those without. These training activities include *conduct software security awareness training* and *deliver on-demand individual training*, which were about 40% and 50% higher in firms with champions than those without, as well as *include security resources in onboarding*, which was 33% higher. Having trained security champions and developers facilitates smarter tool use and more secure development.

Cloud Architecture

Cloud architecture has been around for more than a decade, but like any technology, it continues to change and improve. The combination of modern cloud-native application protection platforms (CNAPP), industry knowledge captured as secure design patterns, and one-click security tooling allows integrating security in ways that were previously more burdensome in company-owned data centers.

In the subset of firms that secure cloud-native architectures via the *implement cloud security controls* activity, we see gains in the Penetration Testing, Compliance & Policy, and Software Environment practices of 35%, 21%, and 44%, respectively. Activities that are made easier in cloud environments also saw growth from BSIMM13 to BSIMM14. The *use application behavior monitoring and diagnostics* activity grew by 64%, and observations of *monitor automated asset creation* grew by 45%. Additionally, observations of *find and publish secure design patterns from the organization*, *require use of approved security features and frameworks*, and *use application containers to support security goals* grew by around 25%. We expect to see this trend continue as firms continue to target cloud environments for new development.

SECURITY ECONOMICS

Not all trends are positive, and many companies have seen reduced security budgets. Activities that rely on experts to perform manual tasks have seen declines as security teams seek to maximize their return on investment by focusing on automation. The *develop an operations software inventory* and *use a data classification scheme for software inventory* activities saw a 9 and 7 count drop in observations for BSIMM14. Additionally, expert-driven tasks like *begin to build and apply adversarial security tests (abuse cases)* declined 25%, *use centralized defect reporting to close the knowledge loop* shrank 18%, and *maintain and use a top N possible attacks list* dropped 31%. The rise of automated activities that allow security teams to shift everywhere and thrive in the cloud appears to be due to a focus in attention away from expensive, slow, and manual security activities.

TOPICS WE'RE WATCHING

This year saw huge changes in priorities, technologies, and possibilities. The demands of cloud, toolchains, tools, application security adjacencies, AI, and government scrutiny are leading to a vastly increased program scope, which is in turn necessitating a new era of shared responsibility between SSGs and engineering.

Participant feedback indicates that the following might influence their future efforts:

- The continuing refinement of the product security culture as commercial software firms seek to meet different security objectives through greater coordination of application, cyber, manufacturing, and IT security teams.
- Regulations as other countries follow the United States government lead and mandate security requirements for any government software suppliers, which will naturally flow downhill to their suppliers.
- The expansion of AI-generated code, integration of machine learning models into software, and use of intelligent bots in the SDLC.

PART 3: BSIMM PARTICIPANTS

THE BSIMM PARTICIPANTS

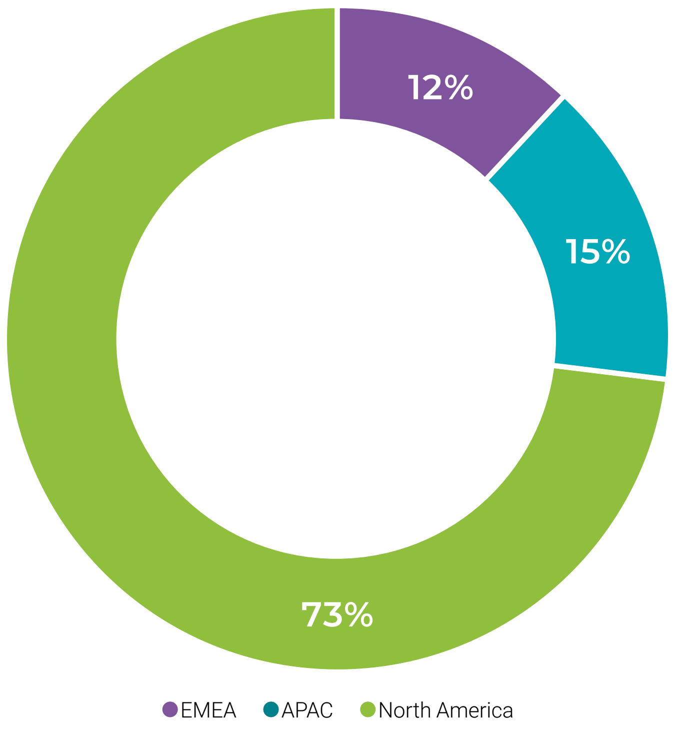
BSIMM participants comprise software security leaders and team members from around the globe. They have a common mission to continuously improve their SSIs in light of changes in the world around them. You can use the information they've provided to learn from their efforts.

This 2023 edition of the BSIMM report—BSIMM14—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 participating organizations fall across various verticals, including cloud, financial services, FinTech, ISVs, insurance, IoT, healthcare, and technology organizations (see Figure 2).

Unique in the software security industry, the BSIMM project has grown from nine participating companies in 2008 to 130 in 2023, currently with nearly 3,600 software security group members and more than 7,500 satellite members (aka security champions). Today, the average age of the participants' SSIs is 5.2 years. As seen in Table 3, the BSIMM project shows consistent growth even as organizations enter and leave over time.



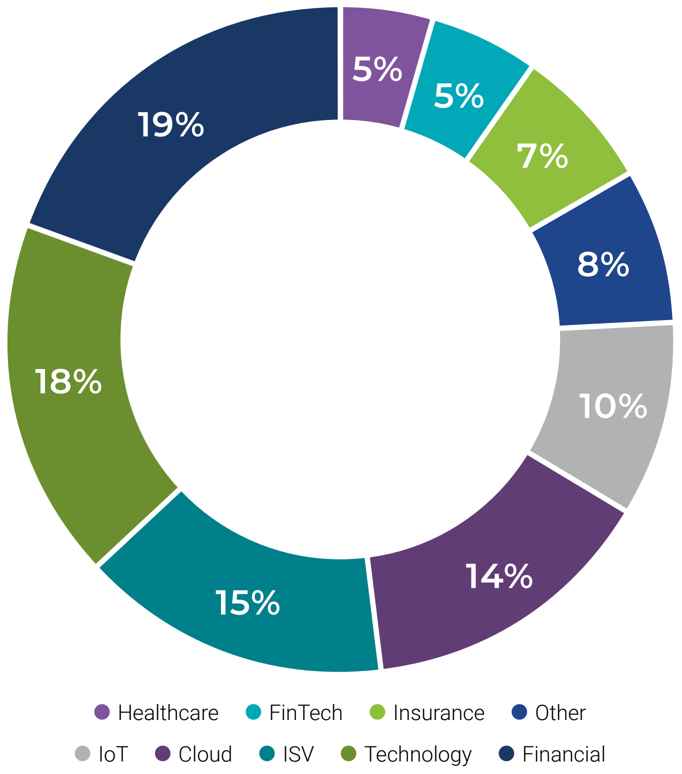


FIGURE 2. BSIMM14 PARTICIPANTS. Participant percentages per tracked region and vertical.

BSIMM PARTICIPANT NUMBERS OVER TIME									
	BSIMM14	BSIMM13	BSIMM12	BSIMM11	BSIMM10	BSIMM9	BSIMM8	BSIMM7	BSIMM1
Firms	130	130	128	130	122	120	109	95	9
SSG Members	3,572	3,342	2,837	1,801	1,596	1,600	1,268	1,111	370
Satellite Members	7,427	8,508	6,448	6,656	6,298	6,291	3,501	3,595	710
Developers	267,731	408,999	398,544	490,167	468,500	415,598	290,582	272,782	67,950
Applications	96,361	145,303	153,519	176,269	173,233	135,881	94,802	87,244	3,970
Average SSG Age (Years)	5.20	5.00	4.41	4.32	4.53	4.13	3.88	3.94	5.32
SSG Average of Averages (SSG per Developers)	3.87 / 100	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 PARTICIPANT NUMBERS OVER TIME. The chart shows how the BSIMM study has grown over the years.

ACKNOWLEDGEMENTS

Our thanks to the 130 executives, including those who wish to remain anonymous, from the SSIs we studied to create BSIMM14.

Our thanks also to the nearly 151 individuals who helped gather the data for the BSIMM data pool over time.

In particular, we thank Adam Brown, Akhil Mittal, Akshay Sawant, Alex Jupp, Alistair Nash, Anders Stadum, Balaji Padmanabhan, Ben Hutchison, Brendan Sheairs, Chandu Ketkar, Daniel Cohen, Devaraj Munuswamy, Don Pollicino, Durai G, Eason Yu, Eli Erlikhman, Harshad Janorkar, Ibrahim Khan, Iman Louis, Jatin Virmani, Jonathan Dunfee, Larrry Cox, Lekshmi Nair, Li Zhao, Matt Chartrand, Michael Fabian, Mike Lyman, Nivedita Murthy, Rajiv Harish, Ravinder Reddy Amireddy, Sachin Shetty, Sam Schueller, Sammy Migues, Smith Kaneria, Stanislav Sivak, Stephen Gardner, Surya Uddhi Nagaraj, Thaddeus Bender, Uzear Ahmed, Vijay Sharma, Warrie Proffitt, and Zhihao Yu. We would also like to thank David Johansson and Surya Uddhi Nagaraj for their work managing the BSIMM tooling and data and creating the extracts used in this report. In addition we would like to thank Austin Kleineschay, Jennifer Stout, and Rachel Bay for their work on various aspects of this report.

BSIMM14 was authored by Jamie Boote, Eli Erlikhman, Ben Hutchison, Mike Lyman, and Sammy Migues

AARP	Honeywell	QlikTech International AB
Aetna	HUMAN Security	Realtek
Airoha	Imperva	Reckitt
AON	Inspur Software	Sammons Financial
Arlo	Intralinks	ServiceNow
Axway	iPipeline	Signify
Bank of America	Johnson & Johnson	SonicWall
Bell Network	Landis+Gyr	Synchrony Financial
CIBC	Lenovo	TD Ameritrade
Citi	MassMutual	Teradata
Depository Trust & Clearing Corporation	MediaTek	Trainline
Diebold Nixdorf	Medtronic	U.S. Bank
Egis Technology	MiTAC	Unisoc
Eli Lilly and Company	Navient	Vanguard
EQ Bank	Navy Federal Credit Union	Veritas
Fidelity	NEC	Verizon Media
Finastra	NetApp	Vivo
Genetec	Oppo	World Wide Technology
HCA Healthcare	Pegasystems	ZoomInfo
	Principal Financial	

PART 4:
QUICK GUIDE TO SSI MATURITY

QUICK GUIDE TO SSI MATURITY

Twelve 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.

A 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 each SSI stakeholder responsible for aspects of software security risk management in your organization.

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

- Do you maintain at least a near-current view of all your software and development assets, including internal code, third-party code, open source, development environments and toolchains, infrastructure-as-code, and other software assets?
- Are you creating and using in your risk management processes SBOMs that detail all the components in the SSI's software portfolio?
- Do you have a near-real-time view of 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, etc.—to align culture, approach, technology stacks, and testing strategies?
- Have you scaled your security champions program across your software portfolio, including skills specific to automation, technology stacks, application architectures, cloud-native development, and other important DevOps needs?
- Are you delivering important security policy, standards, and guidelines as-code that run 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 a secure, auditable, governance-as-code-driven SDLC?
- Are you following a shift everywhere strategy to move from large, time-consuming security tests to smaller, faster, timelier, pipeline-driven security tests conducted to improve engineering team performance?
- Are you managing supply chain risk through vendor software assurance, governance-driven access and usage controls, maintenance standards, and collected provenance data?

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 and automation improvements in your DevOps toolchain or governance improvements in your policies and standards?
- Does your SSI strategy include security efforts needed specifically for modern technologies, such as cloud, container, orchestration, open source management, development pipeline, etc.?
- Are you actively experimenting with new technologies, such as AI and large language models (LLMs),, that have the opportunity to integrate security and engineering functions while also reducing engineering friction?

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 of efforts vs. 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 scaling and 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 and where you want to go in your journey 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 participants, whether as a whole or for specific verticals. A direct comparison of your efforts to the BSIMM14 scorecard for the entire data pool (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 is expected 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 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 (security champions), engineering, QA, operations, and security testing. As needed, extend the stakeholder list to include teams from reliability, cloud, privacy, training, infrastructure, resilience, AI/ML, 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 connected 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 on the intersection of the interviewee's role and specific BSIMM activities. 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 1 or 0, 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 first to your stakeholders' realistic expectations and then also to what's common in the data pool. 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, your executive team, and your Board (with each likely needing different metrics).

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 BSIMM14 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 BSIMM14 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 data pool. If you want to evaluate your scorecard against a particular vertical, refer to Appendix E.

Once you have determined where you stand with activity efforts 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 to 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	BSIMM14 FIRMS (OUT OF 130)	EXAMPLE FIRM	ACTIVITY	BSIMM14 FIRMS (OUT OF 130)	EXAMPLE FIRM	ACTIVITY	BSIMM14 FIRMS (OUT OF 130)	EXAMPLE FIRM	ACTIVITY	BSIMM14 FIRMS (OUT OF 130)	EXAMPLE FIRM
STRATEGY & METRICS			ATTACK MODELS			ARCHITECTURE ANALYSIS			PENETRATION TESTING		
[SM1.1]	101	1	[AM1.2]	73		[AA1.1]	108	1	[PT1.1]	114	
[SM1.3]	80		[AM1.3]	49	1	[AA1.2]	59	1	[PT1.2]	102	1
[SM1.4]	118		[AM1.5]	81		[AA1.4]	63		[PT1.3]	85	1
[SM2.1]	73		[AM2.1]	16		[AA2.1]	35		[PT2.2]	42	
[SM2.2]	71		[AM2.6]	16	1	[AA2.2]	34	1	[PT2.3]	55	
[SM2.3]	71		[AM2.7]	15	1	[AA2.4]	40	1	[PT3.1]	30	1
[SM2.6]	77		[AM2.8]	20		[AA3.1]	20		[PT3.2]	21	
[SM2.7]	62	1	[AM2.9]	16		[AA3.2]	8				
[SM3.1]	32		[AM3.2]	8		[AA3.3]	17				
[SM3.2]	23		[AM3.4]	13							
[SM3.3]	32		[AM3.5]	11							
[SM3.4]	8										
[SM3.5]	0										
COMPLIANCE & POLICY			SECURITY FEATURES & DESIGN			CODE REVIEW			SOFTWARE ENVIRONMENT		
[CP1.1]	103	1	[SFD1.1]	100	1	[CR1.2]	84	1	[SE1.1]	88	
[CP1.2]	114	1	[SFD1.2]	95	1	[CR1.4]	112	1	[SE1.2]	113	1
[CP1.3]	101	1	[SFD2.1]	45		[CR1.5]	74		[SE1.3]	92	1
[CP2.1]	58		[SFD2.2]	70		[CR1.7]	55		[SE2.2]	68	1
[CP2.2]	63		[SFD3.1]	18		[CR2.6]	26	1	[SE2.4]	45	
[CP2.3]	72		[SFD3.2]	22		[CR2.7]	20		[SE2.5]	63	1
[CP2.4]	62		[SFD3.3]	9		[CR2.8]	28	1	[SE2.7]	47	1
[CP2.5]	80	1				[CR3.2]	17		[SE3.2]	18	
[CP3.1]	38					[CR3.3]	5		[SE3.3]	18	
[CP3.2]	34					[CR3.4]	3		[SE3.6]	22	
[CP3.3]	15					[CR3.5]	4		[SE3.8]	2	
									[SE3.9]	0	
TRAINING			STANDARDS & REQUIREMENTS			SECURITY TESTING			CONFIG. MGMT. & VULN. MGMT.		
[T1.1]	76	1	[SR1.1]	94	1	[ST1.1]	110	1	[CMVM1.1]	117	1
[T1.7]	64	1	[SR1.2]	103	1	[ST1.3]	91	1	[CMVM1.2]	95	
[T1.8]	59		[SR1.3]	98		[ST1.4]	62		[CMVM1.3]	98	1
[T2.5]	44		[SR1.5]	101	1	[ST2.4]	23		[CMVM2.1]	92	
[T2.8]	27	1	[SR2.2]	75		[ST2.5]	34		[CMVM2.3]	53	
[T2.9]	32	1	[SR2.5]	63	1	[ST2.6]	25		[CMVM3.1]	14	
[T2.10]	26		[SR2.7]	58		[ST3.3]	16		[CMVM3.2]	24	
[T2.11]	30		[SR3.2]	18		[ST3.4]	4		[CMVM3.3]	18	
[T2.12]	28		[SR3.3]	19		[ST3.5]	3		[CMVM3.4]	30	1
[T3.1]	8		[SR3.4]	21		[ST3.6]	6		[CMVM3.5]	16	1
[T3.2]	14								[CMVM3.6]	3	
[T3.6]	8								[CMVM3.7]	35	
									[CMVM3.8]	0	

FIGURE 3. BSIMM14 EXAMPLEFIRM SCORECARD. A scorecard helps everyone understand the software security efforts that are currently underway. It also helps organizations make comparisons to participants 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 273 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.** 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 cloud, infrastructure, development, tooling, 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.
- **Security champions (satellite).** 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 (and AI!) 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. Product management teams that create non-functional security requirements (NFSRs) greatly improve the ability of QA teams to create security tests.
- **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.
- **GRC, legal, and data privacy.** Specialists can help ensure that regulations, laws, contracts, and client expectations are translated into mandatory program, software, and process security requirements.
- **Procurement and vendor management.** Holistic software security means securing software from vendors and other sources. Dedicated security-aware vendor management and procurement stakeholders have a key role to play in supporting the organization's software supply chain risk management strategy and SSI. Software procurement and vendor managers can help facilitate assurance interactions, including security assessments of vendors, to ensure that acquired and supplied software aligns with organizational security objectives and SSI 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 15 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, making the 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 273 firms so far. In addition, we've conducted assessments for 18 organizations that have rejoined the participant group after aging out. In 44 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, then by adding new activities when warranted; we've added 17 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 BSIMM participants. We organize that core knowledge into an SSF, represented in Figure 4, that is organized into four domains—Governance, Intelligence, SSDL Touchpoints, and Deployment—with those domains containing the 126 BSIMM14 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 participants. Doing this helps 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 data pool 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)	4. Attack Models (AM)	7. Architecture Analysis (AA)	10. Penetration Testing (PT)
2. Compliance & Policy (CP)	5. Security Features & Design (SFD)	8. Code Review (CR)	11. Software Environment (SE)
3. Training (T)	6. Standards & Requirements (SR)	9. Security Testing (ST)	12. Configuration Management & Vulnerability Management (CMVM)

FIGURE 4. THE SOFTWARE SECURITY FRAMEWORK. Twelve practices align with the four high-level domains and contain the 126 BSIMM14 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.		

[SM2.7: 62]
Create evangelism role and perform internal marketing.

Build support for software security throughout the organization via ongoing evangelism and ensure that everyone aligns on security objectives. 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 champion or 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.

GOVERNANCE		
ACTIVITY	BSIMM14 FIRMS (OUT OF 130)	EXAMPLE FIRM
STRATEGY & METRICS		
[SM1.1]	101	1
[SM1.3]	80	
[SM1.4]	118	
[SM2.1]	73	
[SM2.2]	71	
[SM2.3]	71	
[SM2.6]	77	
[SM2.7]	62	1
[SM3.1]	32	
[SM3.2]	23	
[SM3.3]	32	
[SM3.4]	8	
[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—e.g., 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 BSIMM14

☆ New Activity in BSIMM14

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: 101] 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: 80] 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: 118] 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, such as risk-ranking thresholds or defect data, to make a go/no-go decision. Importantly, the conditions need not be enforced at this stage—e.g., the SSG can collect security testing results for each project prior to release, then provide an 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: 71] 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, etc. 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: 71] 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, can 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 delivered via automation (e.g., as-code).

[SM2.6: 77] 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 use SSG-approved criteria to sign off on the state of all software prior to release. The sign-off policy might also require the accountable person to, e.g., 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: 62] Create evangelism role and perform internal marketing.

Build support for software security throughout the organization via ongoing evangelism and ensure that everyone aligns on security objectives. 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 champion or 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: 32] 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: 23] Make SSI efforts part of external marketing.

To build external awareness, the SSG helps market the SSI beyond internal teams. The process of sharing details externally and inviting critique is used to bring new perspectives into the firm. Promoting the SSDL externally can turn security efforts into a market differentiator, and feedback from external marketing can grow an SSI's risk reduction exercises into a competitive advantage. The SSG 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.

[SM3.3: 32] 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 relying on 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: 8] 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 policy-driven access and usage controls, maintenance standards (see [SE3.9]), and captured software provenance data (see [SE2.4]). Apply these processes to external (see [SR2.7]), bespoke, and internally developed software (see [SE3.9]) to help 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 risk, setting organizational software security policy, and auditing against that policy.

[CP1.1: 103] Unify regulatory pressures.

Have a cross-functional team that understands the constraints imposed on software security by regulatory or compliance drivers that are applicable to the organization and its customers. The team takes a common approach that removes redundancy and conflicts to unify 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: 114] 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 privacy 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 and complexities for PII protection.

[CP1.3: 101] Create policy.
The SSG guides the organization by creating or contributing to software security policies that satisfy 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. The policies include a unified approach for satisfying the (potentially lengthy) list of security drivers at the governance level so that 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 this 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) and where it's stored are necessary so that 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 one. System architectures have evolved such that PII will often flow into cloud-based service and endpoint device ecosystems, then come to rest there (e.g., content delivery networks, workflow systems, mobile devices, IoT devices), making it tricky to keep an accurate PII inventory.

[CP2.2: 63] 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, e.g., acknowledge compliance issues that haven't been mitigated or compliance-related SSDL steps that have been skipped, but sign-off 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: 72] Implement and track controls for compliance.
The organization can demonstrate compliance with applicable requirements because its SSDL is aligned with the control statements that were 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 then 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 security and 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 method aren't sufficient here.

[CP2.5: 80] Ensure executive awareness of compliance and privacy obligations.

Gain buy-in around compliance and privacy obligations by providing executives with plain-language explanations of both the organization's compliance and privacy requirements and the potential consequences of failing to meet those requirements. 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: 38] Document a software compliance story.

The SSG can demonstrate the organization's up-to-date software security compliance story on demand. A compliance story is a collection of data, artifacts, policy controls, or other documentation that shows the compliance state of the organization's software and processes. 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: 34] 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: 15] 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 in 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 including KPIs, OKRs, KRIs, SLIs, SLOs, or other organizational metrics can indicate where 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 effectiveness.

Governance: Training (T)

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

[T1.1: 76] 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, security champions, 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 only 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: 64] 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) to achieve its intended purpose. Ineffective (e.g., aged, off-topic) training or training that isn't used won't create any change. Hot engineering topics like containerization and security orchestration, and new training 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: 59] 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: 44] 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: 27] 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, design review, 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: 32] Deliver role-specific advanced curriculum.

Software security training goes beyond building awareness (see [T1.1]) to 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 the students. An organization could offer tracks for its engineers, for example, supplying 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 other 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: 26] 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: 30] 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. While one refresher module can be used for multiple roles (see [T2.9]), coverage of new topics and changes to the previous year's content should result in a significant amount of fresh content.

[T2.12: 28] 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.

[T3.1: 8] 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: 14] Provide training for vendors and outsourced workers.
Vendors and outsourced workers receive appropriate software security training, comparable to the level of 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.

[T3.6: 8] Identify new satellite members (security champions) through observation.
Future satellite members (e.g., security champions) are recruited by noting people who stand out during opportunities that show skill and enthusiasm, such as training courses, office hours, capture-the-flag exercises, hack-a-thons, etc. and 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: 73] 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: 49] Identify potential attackers.
The SSG identifies potential attackers in order to understand and begin documenting 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: 81] 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, etc. 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 everyone 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]). Attack patterns frequently contain details of the targeted asset, attackers, goals, and the techniques used. These resources can be built from scratch or from standard sets, such as the MITRE ATT&CK framework, with the SSG adding 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 both their attack pattern and abuse case creation prioritization and their content over time due to changing software architectures (e.g., zero trust, cloud native, serverless), attackers, and technologies.

[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: 15] Build an internal forum to discuss attacks.

The organization has an internal, interactive forum where the SSG, the satellite (champions), incident response, and others discuss attacks and attack methods. The discussion serves to communicate the attacker perspective to everyone, so 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 communication channel (see [T2.12]) 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 software, 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.

[AM2.8: 20] 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.

[AM2.9: 16] 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 crawling and scraping.

[AM3.2: 8] 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 [AM2.8]) or a disclosing third party could require a new tool, so the SSG, perhaps through the champions, 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, 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]).

[AM3.4: 13] 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 or appropriate SMEs 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., AI, 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.

[AM3.5: 11] Maintain and use a top N possible attacks list.
The SSG periodically digests the ever-growing list of applicable 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."

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: 100] 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 security 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 SDKs, containers, microservices, and APIs. Generic security features often must be tailored for specific platforms. For example, each mobile and cloud platform might 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 and disseminating these defined security features that generates real progress, not simply making a list of them.

[SFD1.2: 95] 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: 45] 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: 70] 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 feature (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: 18] 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, preferably 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 or champions. Review board security guidance can also serve to inform outsourced software providers about security expectations (see [CP3.2]).

[SFD3.2: 22] 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: 9] 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, and 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 and input validation), creating security standards for technologies in use, and creating a standards review process.

[SR1.1: 94] 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 mandate 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. For example, 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: 103] 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 [T2.12]), 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: 98] 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]).

[SR1.5: 101] 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.

[SR2.2: 75] 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). Common triggers for standards review processes include periodic updates, security incidents, major vulnerabilities discovered, adoption of new technologies, acquisition, etc.

[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 SSDF gap analysis (https://csrc.nist.gov/Projects/ssdf), BSIMMsc measurements, or BSIMM scores.

[SR2.7: 58] 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 [SR1.5]). 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 license compliance objectives and 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: 18] Communicate standards to vendors.

Work with vendors to educate them and promote the organization's security standards. A healthy relationship with a vendor often starts with contract language (see [CP2.4]), but 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: 19] 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 (e.g., style), 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: 21] Create standards for technology stacks.
The organization standardizes on the use of specific technology stacks, which 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, configurations, and 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 MEAN 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: 108] 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 collectively 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, etc. 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: 59] Perform design review for high-risk applications.
Perform a design review to determine whether the 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: 63] 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 then use the answers to categorize the application as, e.g., 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: 35] Perform architecture analysis using a defined process.
Define and use a process for 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 more completely 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: 34] Standardize architectural descriptions.
Threat modeling, design review, or AA processes use an agreed-upon format (e.g., diagramming language and icons, not simply a text description) 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: 40] 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, e.g., 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 don't 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]). But even with a good process, consistency is difficult to attain because breaking architecture requires experience, so 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: 8] 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 origin of security flaws, searches for what 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: 17] 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, e.g., 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 [T2.12]), 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 (e.g., developers vs. auditors), manual analysis, and tracking and measuring results.

[CR1.2: 84] Perform opportunistic code review.
Perform code review for high-risk applications in an opportunistic fashion. For example, 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 (see [CR1.4]). 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: 112] 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 judgment, 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 aligned 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: 74] 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: 55] 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). While AI is becoming useful to augment human code review guidance, it likely doesn't have the context necessary to replace it.

[CR2.6: 26] 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 the organization 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. Build a valuable list by using 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 prioritize it 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: 28] Use centralized defect reporting to close the knowledge loop.
The defects found during code review are tracked in a centralized repository that makes it possible to do both summary and trend reporting for the organization. Reported defects drive engineering improvements such as enhancing processes, updating standards, adopting reusable frameworks, etc. For example, 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]) or 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: 17] Build a capability to combine AST results.
Combine application security testing (AST) 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, etc. 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 and identify engineering improvements.

[CR3.3: 5] Create capability to eradicate bugs.

When a security bug is found during code review (see [CR1.2], [CR1.4]), the organization searches for and 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: 3] 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: 4] 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, removing 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 as well as integrating security into standard QA processes. The practice includes the use of opaque-box 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: 110] 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 (e.g., determining what happens when someone enters the wrong password over and over).

[ST1.3: 91] Drive tests with security requirements and security features.

QA targets declarative security mechanisms with tests derived from security requirements and 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, etc., 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: 62] 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, APIs, etc. 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: 23] 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 for them 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: 34] 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: 25] 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 application interfaces and business logic. Test harnesses developed explicitly for specific applications make good places to integrate fuzz testing.

[ST3.3: 16] 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. The point is to measure how broadly the test cases cover the security requirements, which is not the same as measuring how broadly the test cases exercise the code.

[ST3.5: 3] 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 [AM3.5]). This effort turns the corner in QA from testing features to attempting to break the software under test.

[ST3.6: 6] 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, or a push to deployment. Event-driven testing implemented in pipeline automation (rather than testing only in production) 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 typically include 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. In addition to breaking code, this sanity check can also be an effective way to ensure that vulnerability prevention techniques are both used and effective. 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 or Slack) to communicate change requests, but these requests are still tracked explicitly as part of a vulnerability management process.

[PT1.3: 85] Use penetration testing tools internally.

The organization creates an internal penetration testing capability that uses tools as part of an established process. Execution can rest with the SSG or be 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: 42] Penetration testers use all available information.

Penetration testers, whether internal or external, routinely make use of artifacts created throughout the SSDL to do more comprehensive analysis and find more problems. Example artifacts include design documents, architecture analysis results, misuse and abuse cases, code review results, cloud environment and other deployment configurations, and source code if applicable. Focusing on high-risk applications is a good way to start. Note that having access to SSDL artifacts is not the same as using them.

[PT2.3: 55] 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, while 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, etc., deserves some security testing as well.

[PT3.1: 30] 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: 21] Customize penetration testing tools.

Build a capability to create penetration testing tools, or to 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], [AM2.8]) and technology stacks (see [AM3.4]). 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 on 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: 88] 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, RASP or a WAF can do this monitoring, while other kinds of software likely require other approaches, such as custom 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: 113] 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: 92] 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: 68] 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. Software here includes applications, products, scripts, images, firmware, and other forms of code. 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. Where deployment automation is not applicable, customers or deployment teams need installation guides that include hardening guidance and secure configurations.

[SE2.4: 45] 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. Protecting code integrity can also include securing development infrastructure, using permissions and peer review to govern code contributions, and limiting code access to help protect integrity (see [SE3.9]).

[SE2.5: 63] 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, but their planned use can 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: 47] 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: 18] 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 Exception 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: 18] 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., platform-as-a-service), some of this data and the associated predictive analytics might come from a vendor.

[SE3.6: 22] 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, i.e., 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: 2] 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 [SR1.5]), 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 SBOMs. 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]).

[SE3.9: 0] Protect integrity of development toolchains.

The organization ensures the integrity of software it builds and integrates by maintaining and securing all development infrastructure and preventing unauthorized changes to source code and other software lifecycle artifacts. Development infrastructure includes code and artifact repositories, build pipelines, and deployment automation. Secure the development infrastructure by safely handling and storing secrets, following pipeline configuration requirements, patching tools and build environments, limiting access to pipeline settings, and auditing changes to configurations. Preventing unauthorized changes typically includes enforcing least privilege access to code repositories and requiring approval for code commits. Automatically granting access for all project team members isn't sufficient to adequately protect software integrity.

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: 117] 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 prebuilt communication channels with critical vendors (e.g., ISP, monitoring, IaaS, SaaS, PaaS) is also very important.

[CMVM1.2: 95] 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, customer feedback, and telemetry from cloud security posture monitoring, container configuration monitoring, RASP, and similar technologies. 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]).

[CMVM1.3: 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 tracking ability could come in the form of a two-way bridge between defect finders and defect fixers or possibly through intermediaries (e.g., the vulnerability management team), 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.

[CMVM2.1: 92] 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 stakeholders such as 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, a process that has never been used might not actually work.

[CMVM2.3: 53] 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: 14] 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 in operations, 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, addressing 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: 24] 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, the 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 could 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 for 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: 30] 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: 16] 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]), provenance data about 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: 35] 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 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 (see [CMVM3.3]). 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 defects 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, etc., to limit the ability to successfully attack deployed applications. Combining attack intelligence (see [AM1.5]) with information about software assets (see [AM2.9]) 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 an 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 (even when the groups have many shared responsibilities). 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.

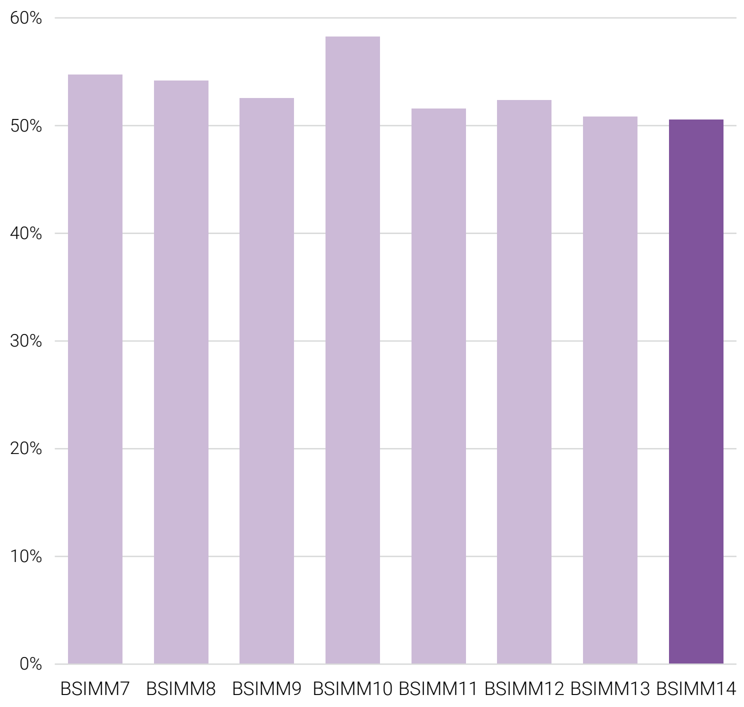


FIGURE 5. PERCENTAGE OF SSGS WITH A CISO AS THEIR NEAREST EXECUTIVE. Assuming new CISOs generally receive responsibilities for SSIs, this data suggests that CISO role creation is also flattening out.

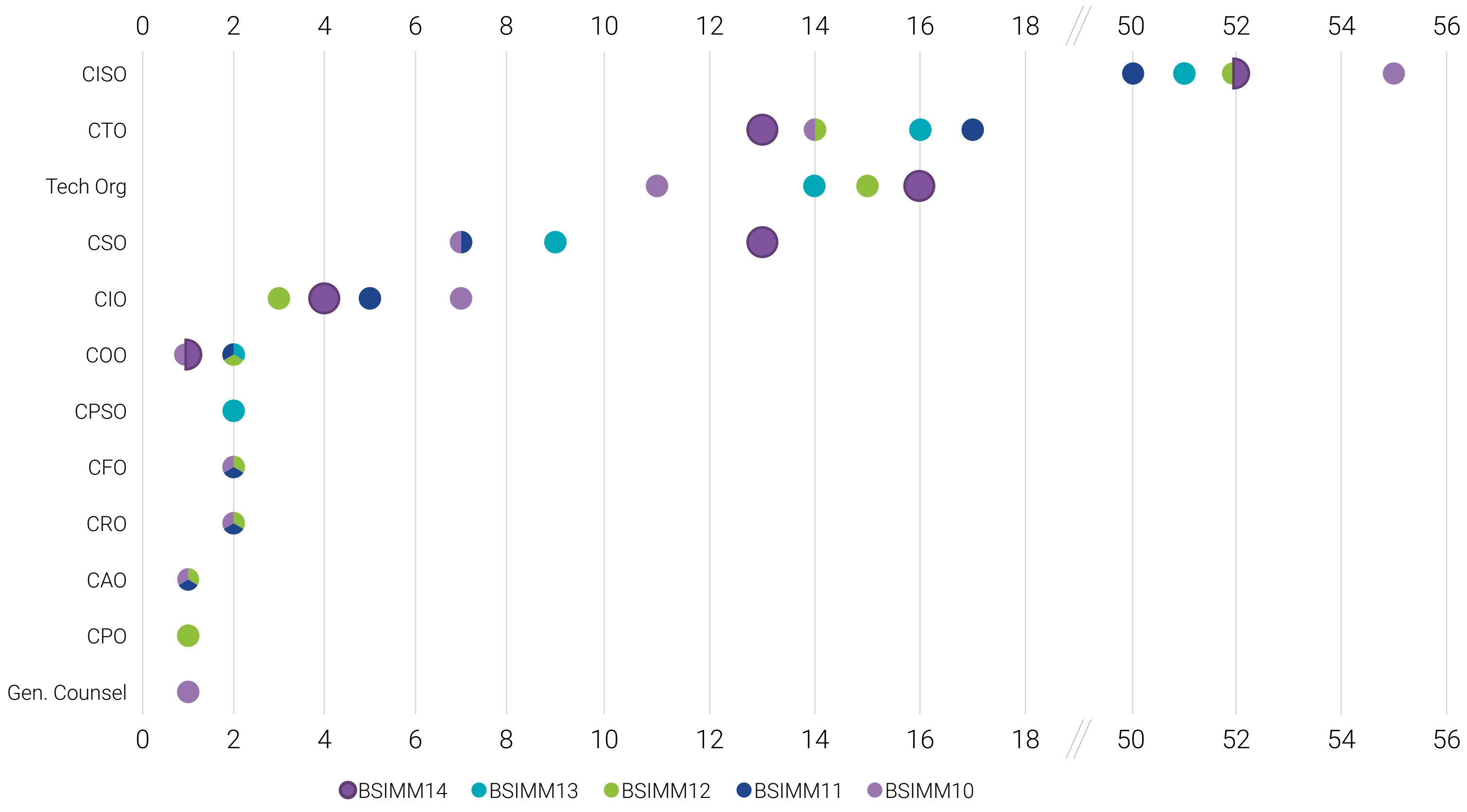


FIGURE 6. NEAREST EXECUTIVE TO SSG. Although many SSGs have a CISO as their nearest executive, we see a variety of executives overseeing software security efforts in the 130 BSIMM14 firms.

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 BSIMM participation 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 BSIMM14, as shown in Figure 6. The larger purple circles show by percentage the SSG leader's nearest executive in the BSIMM14 data pool, while smaller circles show the percentages for previous BSIMMs. For example, a CISO is the closest executive in 52% of organizations (67 of 130) in the BSIMM14 data pool, and that percentage ranged from 50% to 58% in BSIMM7 through BSIMM12. Starting with the BSIMM13 data pool, we no longer see SSGs reporting to CRO (risk), CAO (assurance), CPO (privacy), and General Counsel roles. Note that for BSIMM14, we added 23 firms and removed 23 others, which also affects analysis of reporting chains. Of course, across various organizations, not all people with the same title perform, prioritize, enforce, or otherwise provide resources for the same efforts in the same way.

CISOs in turn report to different executives among the 130 BSIMM14 firms. Figure 7 shows that CISOs report most commonly to CIOs (21 of 67, or almost 32% of the time) and report directly to the CEO about 12% of the time (8 of 67).

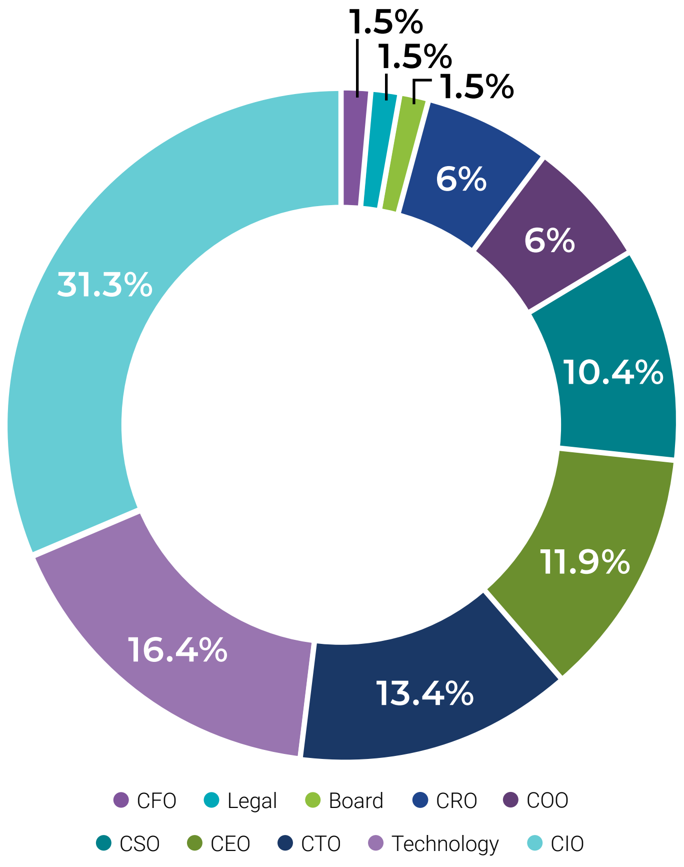


FIGURE 7. TO WHOM THE CISO REPORTS. For the BSIMM14 participants, the CISO reports to a variety of roles, with the most common being the CIO, CTO, and a technology executive (e.g., head of engineering, architecture, or software).

SOFTWARE SECURITY GROUP LEADERS

SSG leaders are individuals in charge of day-to-day efforts in the 130 SSIs we studied for BSIMM14. They have a variety of titles, such as the following:

- Application Security Manager
- Chief Product Officer
- Chief Product Security Officer
- Director Cloud and Application Security
- Director Cybersecurity
- Director Information Security
- Director Product Assurance
- Manager Software Security Office
- Product Security Officer
- Security Director
- Senior Director Security Engineering
- Senior Manager Information Security
- SVP Engineering
- SVP Information and Application Security
- VP Cybersecurity
- VP DevSecOps
- VP Engineering
- VP Information Security
- VP Product Security
- VP Security Architecture
- VP Security Compliance

When the SSG leader is an executive themselves, which happens 12% of the time (15 out of 130), they are CISOs almost 60% of the time (9 out of 15), with other titles being CTO, CPSO (Chief Product Security Officer), and CSO. 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 two hops away from the CEO.

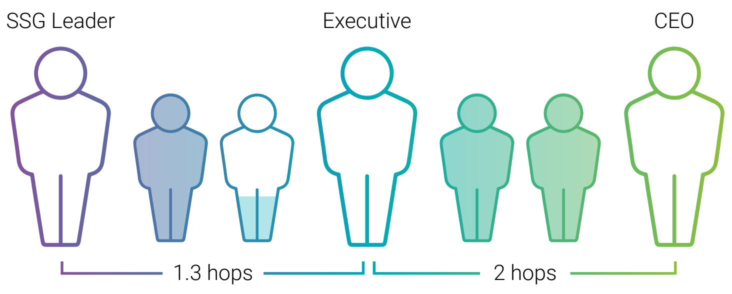


FIGURE 8. SSG LEADERSHIP REPORTING CHAINS. SSG leaders are typically three or four hops away from the CEO.

SOFTWARE SECURITY GROUP (SSG)

Each of the 130 initiatives in BSIMM14 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 such a group 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 SSG-related statistics across the 130 BSIMM14 firms, but note that a large outlier affects the numbers this year. The "Notes" column shows the effect of removing outliers, or the top 10 firms, for that SSG characteristic. 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. Refer to Appendix H for more details on how SSGs evolve over time.

SECURITY CHAMPIONS (SATELLITE)

In addition to the SSG, many SSIs have identified individuals (often developers, testers, architects, cloud and DevOps engineers, and other SDLC roles) who are a driving force in improving software security but are (likely) not directly employed in the SSG. We historically refer to this group as the satellite, while many organizations today refer to them 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 champions, as shown in Figure 9. Having security champions carry out software security activities removes SSG members from the engineering critical path and empowers engineering teams to own their software security deliverables and share responsibility for software security objectives.

Security champions 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 champions proactively identify potential frictions and minimize them.

A successful satellite gets 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, tools, and security features, rather than, e.g., 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.

THE SOFTWARE SECURITY GROUP					
STATISTICS	AVERAGE	MEDIAN	LARGEST	SMALLEST	NOTES – NO OUTLIERS
SSG Size	27.1	8.5	892.0	1.0	Average drops to 20.4 (one outlier)
SSG Members to Developer Ratio (per 100 Developers)	3.87	1.38	51.43	0.02	Average drops to 2.09 (no top 10)
SSG to Developer Ratio (700+ Developers) - 66 Firms (per 100 Developers)	1.61	0.69	14.87	0.02	
SSG to Developer Ratio (Less than 700 Developers) - 64 Firms (per 100 Developers)	6.19	2.33	51.43	0.33	
Number of Developers	2,059	700	30,000	25	
Number of Applications	741.24	121.00	8000.00	1.00	
SSG Age	5.20	4.50	23.00	0.10	
Satellite to Developer Ratio (per 100 Developers)	5.57	1.74	102.20	0.00	Average drops to 4.82 (one outlier)
Satellite to Developer Ratio (700+ Developers) - 66 Firms (per 100 Developers)	4.75	2.00	57.14	0.00	
Satellite to Developer Ratio (Less than 700 Developers) - 64 Firms (per 100 Developers)	6.43	0.10	102.20	0.00	
SSG to Application Ratio (per 100 Developers)	81.81	8.79	2000.00	0.07	Average drops to 51.84 (one outlier)

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 each participating firm. In the Notes column, we show the impact of removing outliers in the data.

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.
- GRC, legal, and 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 contract terms, regulatory and compliance requirements including privacy regulations, definition of privacy requirements, and tracking of PII and other regulated data categories. This has become increasingly common in response to requirements such as GDPR, CCPA, and other regulations.
- Procurement and vendor management need to communicate and enforce security requirements with vendors, including those who supply on-premises products, custom software, and SaaS. Software supply chain vendors are increasingly subjected to software security SLAs and reviews (such as the PCI SSF and the Secure Software Development Framework [SSDF]) 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. Procurement and vendor management play a vital role but aren't the only stakeholders responsible for managing software supply chain risk.

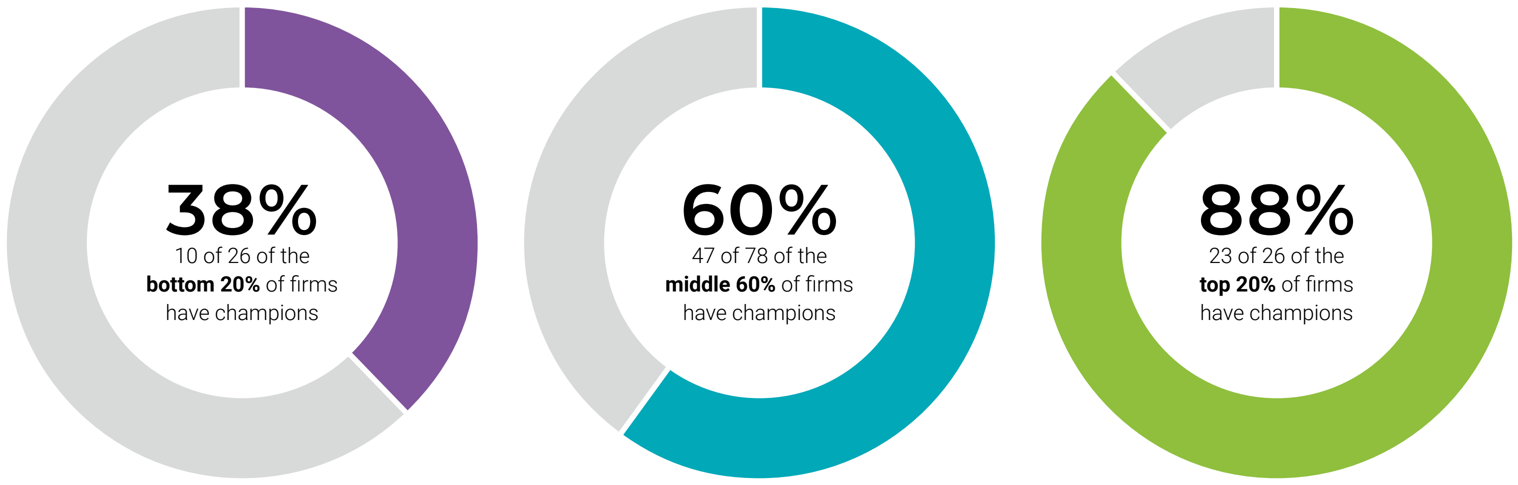


FIGURE 9. THE SATELLITE AND THE BSIMM SCORE. Eighty-eight percent of the top-scoring firms in the BSIMM14 data pool have a satellite (security champions). In contrast, fewer than 40% of bottom-scoring firms have one.

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 or 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, chosen foundational activities (e.g., those observed most frequently in the data pool), acquired some resources, and created 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 efficiently 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 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, along with 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 protection of 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. 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 or 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 the enabling stage. There is also constant change in tools, skill levels, external expectations, attackers, attacks, resources, culture, and everything else. You can use the BSIMM14 participants scorecard (see Figure 17 in Appendix D) 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.

CONSTRUCTION LESSONS FROM THE PARTICIPANTS

The purpose of the BSIMM is to measure SSIs. While the BSIMM doesn't directly measure SSI architecture, evolution, or motivations, our experience with more than 273 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.

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 participants:

- 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 toolchain 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.

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.

Regardless of its origin point, each culture usually arrived at an SSI 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 of years old are driven top-down by governance objectives, even those started by engineering for engineering. Evangelism, peer pressure, and local

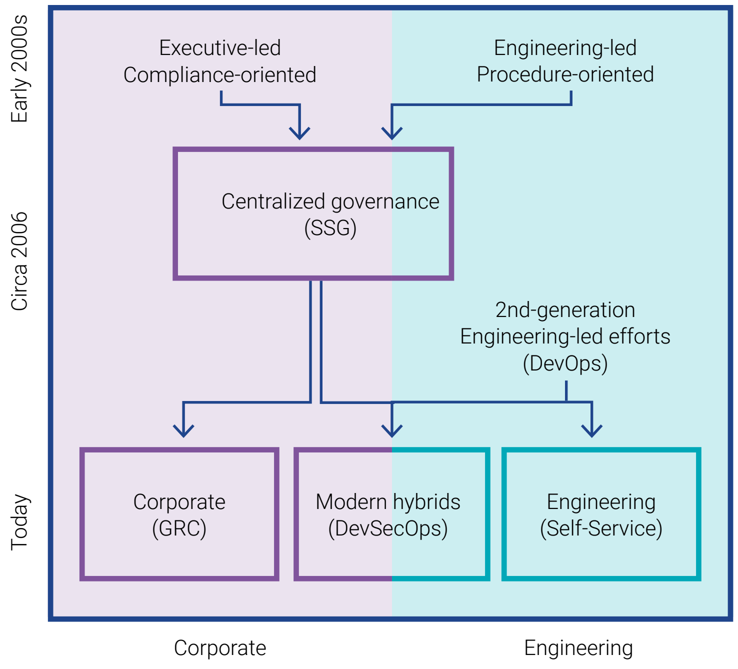

Early 2000s
Circa 2006
Today
Executive-led Compliance-oriented
Engineering-led Procedure-oriented
Centralized governance (SSG)
2nd-generation Engineering-led efforts (DevOps)
Corporate (GRC)
Modern hybrids (DevSecOps)
Engineering (Self-Service)
Corporate
Engineering

FIGURE 10. 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.

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 approaches 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 10 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.

Understanding More About DevOps

The DevOps movement has highlighted the tensions between established SSIs and engineering efforts that address 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 OSS in applications but routinely downloading many other open source packages to build and manage software and processes, etc. 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 that 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 competing 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.

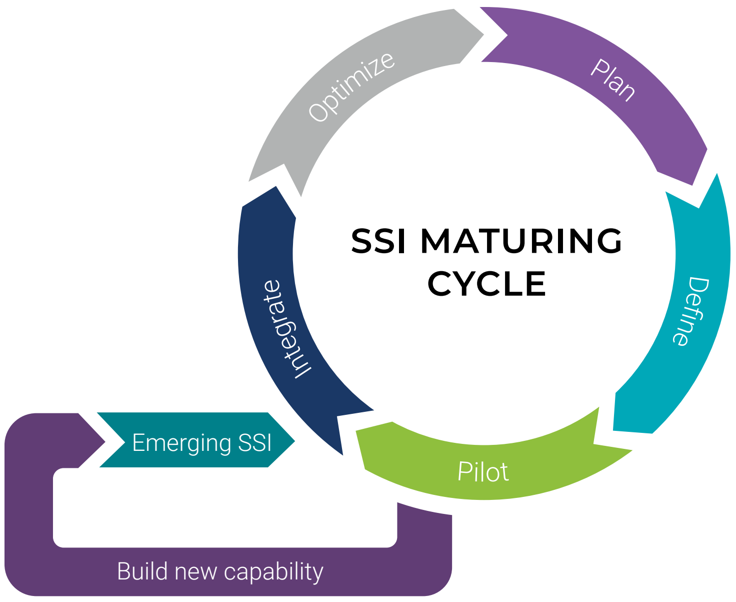


FIGURE 11. MOVING FROM EMERGING TO MATURING. Building an emerging SSI usually focuses on collecting activities into a single program. Moving from emerging to maturing requires ongoing iterative improvements and expansions. Piloting new capabilities (e.g., security champions or software supply chain risk management) likely requires reapplying the emerging approach for a specific set of activities.

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, yet also 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 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 participants, 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.

FOR AN EMERGING SSI: SDLC TO SSDL

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, AST, and processes for working with stakeholders. Provided below are actionable steps for consolidating an ad hoc effort into an emerging SSI. Keep in mind that most SSIs are multiyear efforts with real budget, mandate, and ownership behind them, though. In addition, while all initiatives look different and are tailored to fit a particular organization, most initiatives share common core activities (see Table 7 in Appendix D).

Figure 12 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 12 includes some activities that have a high impact for emerging SSIs even though they appear to be rarely observed in the BSIMM data pool. 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

PHASE	Create a Software Security Group	Document and Socialize the SSDL	Inventory Applications in the SSG's Purview	Apply Infrastructure Security in Software Environments	Deploy Defect Discovery for High-Priority Applications	Manage Discovered Defects	Publish and Promote the Process
GOVERNANCE	CP1.1	SM1.1 SM2.7 AM3.5 CR2.7		SE2.2		CMVM1.3	SM1.4 SM3.4 CP1.3 SR1.1
ENABLEMENT	T1.1 SFD1.1	SR1.2	AM1.2 CP2.1 CMVM2.3		AA1.4		SM1.3 SR1.2
FLAW AND DEFECT DISCOVERY	SFD1.2		SR1.5		AA1.1 CR1.4 SR1.5 ST1.4 PT1.1 CMVM3.4	CR1.4 PT1.2	ST3.6
OPERATIONS			CMVM1.1	SE1.2 SE1.3 SE2.7		CMVM1.2	
	🟣 🔵 🟢	🟣🟣🟣 🔵 🟢🟢	🟣 🔵🔵 🟢🟢	🟣🟣 🔵🔵 🟢🟢	🟣🟣 🔵🔵🔵 🟢🟢	🟣🟣 🔵 🟢🟢🟢	🟣🟣 🔵🔵 🟢🟢🟢

🟣 People 🔵 Budget 🟢 Duration →

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 12. 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.

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 federated 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 BSIMM14 data shows that SSIs having an age of 18 months or less at the time of assessment (22 of 130 firms) have an average score of about 33.

Following are some details on the steps shown in Figure 12. The included activity 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 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 GRC channels to complement existing IT security standards or create those channels as necessary to secure the SDLC. The SSG can also create a security portal (e.g., website or wiki) that houses SSDL information centrally [SR1.2]. Similar to the approach for prioritizing defect discovery efforts by categorizing attacks and bugs [AM3.5, CR2.7], we observe these emerging SSIs driving initial standards creation from industry top risks, leveraging general sources such as MITRE, ISO, and NIST to form baseline requirements.

Checklist for emerging SSIs

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. **Manage discovered defects.** Resolving issues reduces risk.
7. **Publish and promote.** Roll out the secure SDLC and promote it both bottom-up and top-down.

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

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 [SR1.5]). 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 OSS. 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 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

Bad infrastructure security can undermine good software security, which means the SSG must ensure that 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

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.

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 [SR1.5]. 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.

Manage Discovered Defects

Unaddressed security defects are unmanaged risks. At first, there will be a large backlog of discovered security defects that will have to be bundled and passed through the risk exception process and prioritized into the development backlog. After resolving the technical debt, the ongoing defect management process should be designed to deal with security defects as they are introduced to prevent their release into production systems.

When security defects are discovered, it is the responsibility of the SSI to make sure they are logged and tracked through to completion [CMVM1.3]. Security defects can come from diverse sources, including penetration testers [PT1.2], security tooling [CR1.4], and operations [CMVM1.2] and ideally are logged in a single source of truth for tracking purposes.

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 by 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.

FOR A MATURING 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 12) and is maturing specific aspects of its initiative. That maturing includes both adding new activities and scaling existing ones (see Figure 11). 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, etc., 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.

Checklist for maturing SSIs

1. **Unify structure and consolidate efforts.** Formalize organization, 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 replan multiple times.

Unify Structure and Consolidate Efforts

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), so 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, SR1.5], 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 (e.g., testing) controls to maximize positive impacts 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 can 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-facing 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 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:

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

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 [AM2.9] to overcome limitations in a cloud provider's ability to meet desired policy.

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 ([AM3.5, 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.

FOR AN ENABLING SSI: DATA-DRIVEN IMPROVEMENTS

Achieving software security scale—of expertise, portfolio coverage, tool integration, vulnerability discovery accuracy, process consistency, etc.—remains a top priority. However, firms often scale one or two capabilities (e.g., defect discovery, training) but fail to scale others (e.g., AA, vendor management). Given mature activities, there's a treasure trove of data to be harvested and included in KPI and KRI reporting dashboards. But 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, 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 on 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 in QA might leverage AA 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.

Achieving software security scale—of expertise, portfolio coverage, tool integration, vulnerability discovery accuracy, process consistency, etc.—remains a top priority.

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 BSIMM14.

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 judgments. 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 among the BSIMM participants.

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.

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 BSIMM14 FROM BSIMM13

BSIMM14 includes updated activity descriptions, data from firms in multiple vertical markets, and a longitudinal study. For BSIMM14, we added 23 firms and removed 23, resulting in a data pool of 130 firms. In addition, in the time since we launched BSIMM13, 8 firms conducted reassessments to update their scorecards, and we assessed additional business units for two firms.

As shown below, we used the resulting observation counts to refine activity placement in the framework, which resulted in moving seven activities to different levels. In addition, we added one newly observed activity, resulting in a total of 126 activities in BSIMM14:

- [T3.5] Provide expertise via open collaboration channels became [T2.12]
- [AM2.2] Create technology-specific attack patterns became [AM3.4]
- [AM2.5] Maintain and use a top N possible attacks list became [AM3.5]
- [AM3.1] Have a research group that develops new attack methods became [AM2.8]
- [AM3.3] Monitor automated asset creation became [AM2.9]
- [SR2.4] Identify open source became [SR1.5]
- [CMVM2.2] Track software defects found in operations through the fix process became [CMVM1.3]
- [SE3.9] Protect integrity of development toolchains was added to the model

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.	[T2.12]	Provide expertise via open collaboration channels.
[SM3.2]	Make SSI efforts part of external marketing.	[CP3.2]	Ensure compatible vendor policies.	[T3.1]	Reward progression through curriculum.
[SM3.3]	Identify metrics and use them to drive resourcing.	[CP3.3]	Drive feedback from software lifecycle data back to policy.	[T3.2]	Provide training for vendors and outsourced workers.
[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.	[SR1.5]	Identify open source.
[AM2.6]	Collect and publish attack stories.	[SFD3.1]	Form a review board to approve and maintain secure design patterns.	[SR2.2]	Create a standards review process.
[AM2.7]	Build an internal forum to discuss attacks.	[SFD3.2]	Require use of approved security features and frameworks.	[SR2.5]	Create SLA boilerplate.
[AM2.8]	Have a research group that develops new attack methods.	[SFD3.3]	Find and publish secure design patterns from the organization.	[SR2.7]	Control open source risk.
[AM2.9]	Monitor automated asset creation.			[SR3.2]	Communicate standards to vendors.
[AM3.2]	Create and use automation to mimic attackers.			[SR3.3]	Use secure coding standards.
[AM3.4]	Create technology-specific attack patterns.			[SR3.4]	Create standards for technology stacks.
[AM3.5]	Maintain and use a top N possible attacks list.				

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 design 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 engineering.
[PT1.3]	Use penetration testing tools internally.	[SE1.3]	Implement cloud security controls.	[CMVM1.3]	Track software defects found in operations through the fix process.
[PT2.2]	Penetration testers use all available information.	[SE2.2]	Define secure deployment parameters and configurations.	[CMVM2.1]	Have emergency response.
[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 defects 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 defects 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.
		[SE3.9]	Protect integrity of development toolchains.	[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 126 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 had 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.

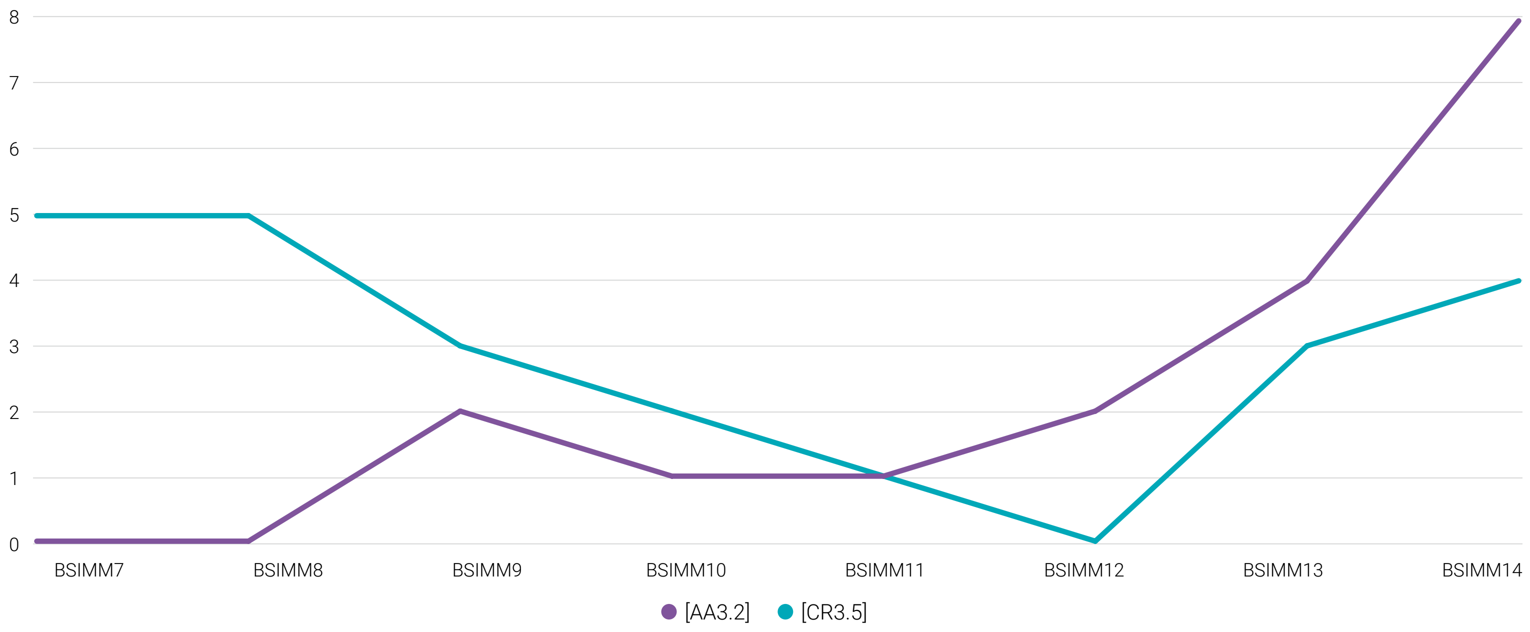


FIGURE 14. NUMBER OF OBSERVATIONS FOR [AA3.2] AND [CR3.5] OVER TIME. [AA3.2 Drive analysis results into standard design patterns] had zero observations during BSIMM7 and BSIMM8, while [CR3.5 Enforce secure coding standards] decreased to zero observations from BSIMM8 to BSIMM12 (the number of observations increased back to four in BSIMM14). Currently, there are three activities with zero observations, one of which was added in BSIMM14.

OBSERVATIONS								
ACTIVITY	BSIMM7	BSIMM8	BSIMM9	BSIMM10	BSIMM11	BSIMM12	BSIMM13	BSIMM14
SE3.4 (now SE2.5)	0	4	11	14	31	44	52	63
SE3.5 (now SE2.7)			0	5	22	33	42	47
SE3.6			0	3	12	14	18	22
SE3.7 (now SE1.3)			0	9	36	59	79	92
SM3.4				0	1	6	5	8
AM3.3 (now AM2.9)				0	4	6	11	17
CMVM3.5				0	8	10	13	16
ST3.6					0	2	3	6
CMVM3.6					0	0	3	3
CMVM3.7						0	20	35
SM3.5							0	0
SE3.8							0	2
CMVM3.8							0	0
SE3.9								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 eight of 130 firms in BSIMM14
- [SFD3.3 Find and publish secure design patterns from the organization] observed in 14 of 51 firms in BSIMM4 but only in nine of 130 firms in BSIMM14
- [SR3.3 Use secure coding standards] observed in 23 of 78 firms in BSIMM6 but only in 19 of 130 firms in BSIMM14

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, 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.

CHANGES FOR BSIMM14 (126 ACTIVITIES)	• [T3.5] Provide expertise via open collaboration channels became [T2.12] • [AM2.2] Create technology-specific attack patterns became [AM3.4] • [AM2.5] Maintain and use a top N possible attacks list became [AM3.5] • [AM3.1] Have a research group that develops new attack methods became [AM2.8] • [AM3.3] Monitor automated asset creation became [AM2.9] • [SR2.4] Identify open source became [SR1.5] • [CMVM2.2] Track software defects found in operations through the fix process became [CMVM1.3] • [SE3.9] Protect integrity of development toolchains added to the model
CHANGES FOR BSIMM13 (125 ACTIVITIES)	• 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

CHANGES FOR BSIMM12 (122 ACTIVITIES)	• 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
CHANGES FOR BSIMM11 (121 ACTIVITIES)	• 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)	• 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)	• 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)	• 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)	• 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 the model
CHANGES FOR BSIMM6 (112 ACTIVITIES)	• 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)	• 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 the model

CHANGES FOR BSIMM4 (111 ACTIVITIES)	• 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 the model • CMVM3.3 Simulate software crises added to the model
CHANGES FOR BSIMM3 (109 ACTIVITIES)	• 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)	• 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)	• 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: BSIMM14

Every organization wants to do software security more effectively and efficiently. You can use this information to understand what the BSIMM participants are 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 76 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 BSIMM14 data.

AGE-BASED PROGRAM CHANGES

Figure 16 shows the distribution of scores among the population compared to Training, Attack Models, and Security Testing 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.

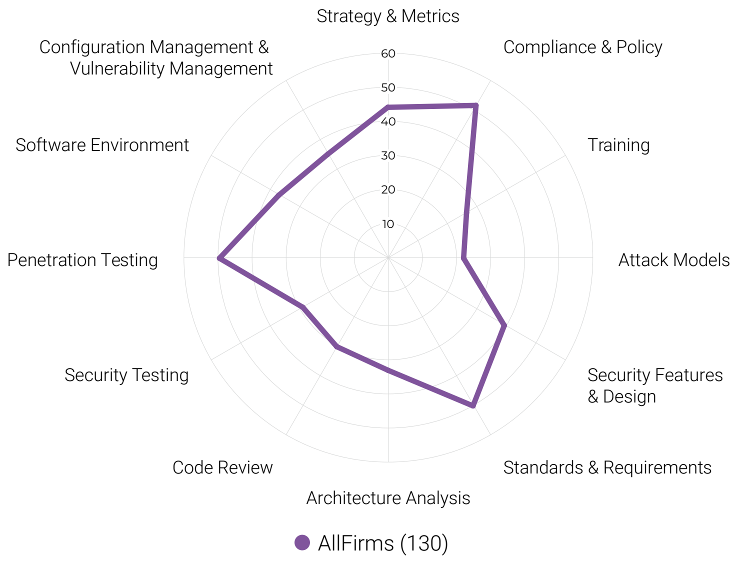


FIGURE 15. ALLFIRMS SPIDER CHART. This diagram shows the average percentage of normalized observations collectively reached in each practice by the 130 BSIMM14 firms. Across these firms, the normalized observations are higher in the Compliance & Policy, Standards & Requirements, and Penetration Testing practices compared to Training, Attack Models, and Security Testing.

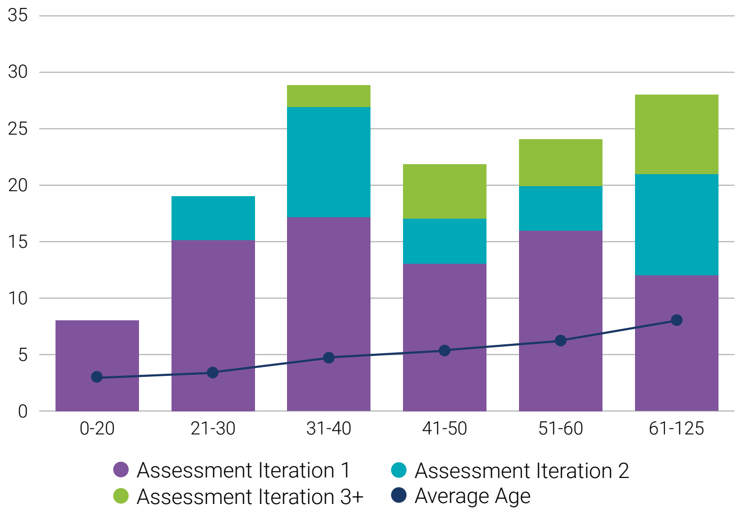


FIGURE 16. BSIMM SCORE DISTRIBUTION. Assessment scores most frequently fall into the 31 to 40 range in BSIMM14, vs. 41 to 50 in BSIMM13 (not shown), with an average SSG age of 4.4 years. In general, firms that mature and continue to use the BSIMM as a measurement tool over time (e.g., round 2, round 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	BSIMM14 FIRMS (OUT OF 130)	BSIMM14 FIRMS (PERCENTAGE)	ACTIVITY	BSIMM14 FIRMS (OUT OF 130)	BSIMM14 FIRMS (PERCENTAGE)	ACTIVITY	BSIMM14 FIRMS (OUT OF 130)	BSIMM14 FIRMS (PERCENTAGE)	ACTIVITY	BSIMM14 FIRMS (OUT OF 130)	BSIMM14 FIRMS (PERCENTAGE)
STRATEGY & METRICS			ATTACK MODELS			ARCHITECTURE ANALYSIS			PENETRATION TESTING		
[SM1.1]	101	77.69%	[AM1.2]	73	56.15%	[AA1.1]	108	83.08%	[PT1.1]	114	87.69%
[SM1.3]	80	61.54%	[AM1.3]	49	37.69%	[AA1.2]	59	45.38%	[PT1.2]	102	78.46%
[SM1.4]	118	90.77%	[AM1.5]	81	62.31%	[AA1.4]	63	48.46%	[PT1.3]	85	65.38%
[SM2.1]	73	56.15%	[AM2.1]	16	12.31%	[AA2.1]	35	26.92%	[PT2.2]	42	32.31%
[SM2.2]	71	54.62%	[AM2.6]	16	12.31%	[AA2.2]	34	26.15%	[PT2.3]	55	42.31%
[SM2.3]	71	54.62%	[AM2.7]	15	11.54%	[AA2.4]	40	30.77%	[PT3.1]	30	23.08%
[SM2.6]	77	59.23%	[AM2.8]	20	15.38%	[AA3.1]	20	15.38%	[PT3.2]	21	16.15%
[SM2.7]	62	47.69%	[AM2.9]	16	12.31%	[AA3.2]	8	6.15%			
[SM3.1]	32	24.62%	[AM3.2]	8	6.15%	[AA3.3]	17	13.08%			
[SM3.2]	23	17.69%	[AM3.4]	13	10.00%						
[SM3.3]	32	24.62%	[AM3.5]	11	8.46%						
[SM3.4]	8	6.15%									
[SM3.5]	0	0.00%									
COMPLIANCE & POLICY			SECURITY FEATURES & DESIGN			CODE REVIEW			SOFTWARE ENVIRONMENT		
[CP1.1]	103	79.23%	[SFD1.1]	100	76.92%	[CR1.2]	84	64.62%	[SE1.1]	88	67.69%
[CP1.2]	114	87.69%	[SFD1.2]	95	73.08%	[CR1.4]	112	86.15%	[SE1.2]	113	86.92%
[CP1.3]	101	77.69%	[SFD2.1]	45	34.62%	[CR1.5]	74	56.92%	[SE1.3]	92	70.77%
[CP2.1]	58	44.62%	[SFD2.2]	70	53.85%	[CR1.7]	55	42.31%	[SE2.2]	68	52.31%
[CP2.2]	63	48.46%	[SFD3.1]	18	13.85%	[CR2.6]	26	20.00%	[SE2.4]	45	34.62%
[CP2.3]	72	55.38%	[SFD3.2]	22	16.92%	[CR2.7]	20	15.38%	[SE2.5]	63	48.46%
[CP2.4]	62	47.69%	[SFD3.3]	9	6.92%	[CR2.8]	28	21.54%	[SE2.7]	47	36.15%
[CP2.5]	80	61.54%				[CR3.2]	17	13.08%	[SE3.2]	18	13.85%
[CP3.1]	38	29.23%				[CR3.3]	5	3.85%	[SE3.3]	18	13.85%
[CP3.2]	34	26.15%				[CR3.4]	3	2.31%	[SE3.6]	22	16.92%
[CP3.3]	15	11.54%				[CR3.5]	4	3.08%	[SE3.8]	2	1.54%
									[SE3.9]	0	0.00%
TRAINING			STANDARDS & REQUIREMENTS			SECURITY TESTING			CONFIG. MGMT. & VULN. MGMT.		
[T1.1]	76	58.46%	[SR1.1]	94	72.31%	[ST1.1]	110	84.62%	[CMVM1.1]	117	90.00%
[T1.7]	64	49.23%	[SR1.2]	103	79.23%	[ST1.3]	91	70.00%	[CMVM1.2]	95	73.08%
[T1.8]	59	45.38%	[SR1.3]	98	75.38%	[ST1.4]	62	47.69%	[CMVM1.3]	98	75.38%
[T2.5]	44	33.85%	[SR1.5]	101	77.69%	[ST2.4]	23	17.69%	[CMVM2.1]	92	70.77%
[T2.8]	27	20.77%	[SR2.2]	75	57.69%	[ST2.5]	34	26.15%	[CMVM2.3]	53	40.77%
[T2.9]	32	24.62%	[SR2.5]	63	48.46%	[ST2.6]	25	19.23%	[CMVM3.1]	14	10.77%
[T2.10]	26	20.00%	[SR2.7]	58	44.62%	[ST3.3]	16	12.31%	[CMVM3.2]	24	18.46%
[T2.11]	30	23.08%	[SR3.2]	18	13.85%	[ST3.4]	4	3.08%	[CMVM3.3]	18	13.85%
[T2.12]	28	21.54%	[SR3.3]	19	14.62%	[ST3.5]	3	2.31%	[CMVM3.4]	30	23.08%
[T3.1]	8	6.15%	[SR3.4]	21	16.15%	[ST3.6]	6	4.62%	[CMVM3.5]	16	12.31%
[T3.2]	14	10.77%							[CMVM3.6]	3	2.31%
[T3.6]	8	6.15%							[CMVM3.7]	35	26.92%
									[CMVM3.8]	0	0.00%

FIGURE 17. BSIMM14 SCORECARD. This scorecard shows how often we observed each of the BSIMM14 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 8 lists the 20 most observed activities in the BSIMM14 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 90.8% of the 130 BSIMM14 participant organizations.

Instead of the top 20 activities overall, Table 7 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, its 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 BSIMM13 and BSIMM14. Level 1 BSIMM activities are the most common activities observed in each practice, and in BSIMM14, seven of the level 1 activities saw double-digit growth despite already being very common. Table 9 shows the top three of these activities.

BSIMM14 TOP ACTIVITIES BY PRACTICE		
ACTIVITY	PERCENTAGE	DESCRIPTION
[SM1.4]	90.8%	Implement security checkpoints and associated governance.
[CP1.2]	87.7%	Identify privacy obligations.
[T1.1]	58.5%	Conduct software security awareness training.
[AM1.5]	62.3%	Gather and use attack intelligence.
[SFD1.1]	76.9%	Integrate and deliver security features.
[SR1.2]	79.2%	Create a security portal.
[AA1.1]	83.1%	Perform security feature review.
[CR1.4]	86.2%	Use automated code review tools.
[ST1.1]	84.6%	Perform edge/boundary value condition testing during QA.
[PT1.1]	87.7%	Use external penetration testers to find problems.
[SE1.2]	86.9%	Ensure host and network security basics are in place.
[CMVM1.1]	90.0%	Create or interface with incident response.

TABLE 7. 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.

BSIMM14 TOP 20 ACTIVITIES BY OBSERVATION PERCENTAGE		
ACTIVITY	PERCENTAGE	DESCRIPTION
[SM1.4]	90.8%	Implement security checkpoints and associated governance.
[CMVM1.1]	90.0%	Create or interface with incident response.
[CP1.2]	87.7%	Identify privacy obligations.
[PT1.1]	87.7%	Use external penetration testers to find problems.
[SE1.2]	86.9%	Ensure host and network security basics are in place.
[CR1.4]	86.2%	Use automated code review tools.
[ST1.1]	84.6%	Perform edge/boundary value condition testing during QA.
[AA1.1]	83.1%	Perform security feature review.
[CP1.1]	79.2%	Unify regulatory pressures.
[SR1.2]	79.2%	Create a security portal.
[PT1.2]	78.5%	Feed results to the defect management and mitigation system.
[CP1.3]	77.7%	Create policy.
[SM1.1]	77.7%	Publish process and evolve as necessary.
[SR1.5]	77.7%	Identify open source.
[SFD1.1]	76.9%	Integrate and deliver security features.
[CMVM1.3]	75.4%	Track software defects found in operations through the fix process.
[SR1.3]	75.4%	Translate compliance constraints to requirements.
[CMVM1.2]	73.1%	Identify software defects found in operations monitoring and feed them back to engineering.
[SFD1.2]	73.1%	Application architecture teams engage with the SSG.
[SR1.1]	72.3%	Create security standards.

TABLE 8. TOP 20 ACTIVITIES BY OBSERVATION PERCENTAGE. Shown here are the most observed activities in the BSIMM14 data pool of 130 firms. This frequent observation means that each activity has broad applicability across a wide variety of SSIs.

Tables 8 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 BSIMM13 and BSIMM14 is to look for trends, such as a high growth in observation rates among common controls. There were 28 activities in BSIMM13 with observations in the range of 40 to 79. The observation rate for six of these activities, shown in Table 10, grew at 16% or higher. In addition, there were 26 activities with observations in the 20 to 39 range, and five of them grew at 20% 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 86% 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]. There 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 (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.

BSIMM14 HIGH-GROWTH ACTIVITIES (1)		
ACTIVITY	GROWTH	DESCRIPTION
[CR1.5]	19.4%	Make code review mandatory for all projects.
[AM1.3]	16.7%	Identify potential attackers.
[SE1.3]	16.5%	Implement cloud security controls.

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.

BSIMM14 HIGH-GROWTH ACTIVITIES (2)		
ACTIVITY	GROWTH	DESCRIPTION
[PT2.3]	22.2%	Schedule periodic penetration tests for application coverage.
[SE2.5]	21.2%	Use application containers to support security goals.
[CR1.5]	19.4%	Make code review mandatory for all projects.
[SE2.2]	19.3%	Define secure deployment parameters and configurations.
[AM1.3]	16.7%	Identify potential attackers.
[SE1.3]	16.5%	Implement cloud security controls.

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.

BSIMM14 HIGH-GROWTH ACTIVITIES (3)		
ACTIVITY	GROWTH	DESCRIPTION
[PT2.3]	22.2%	Schedule periodic penetration tests for application coverage.
[SE2.5]	21.2%	Use application containers to support security goals.
[CR1.5]	19.4%	Make code review mandatory for all projects.
[SE2.2]	19.3%	Define secure deployment parameters and configurations.
[AM1.3]	16.7%	Identify potential attackers.
[SE1.3]	16.5%	Implement cloud security controls.

TABLE 11. ACTIVITIES WITH HIGH GROWTH IN OBSERVATION RATES. This table shows potential new trends in the BSIMM14 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 vs. 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.

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.

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

BSIMM VERTICAL PARTCIPANTS OVER TIME								
	FINANCIAL	FINTECH	ISV	TECH	HEALTHCARE	INTERNET OF THINGS	CLOUD	INSURANCE
BSIMM14	43	12	33	39	10	21	32	15
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 data pool 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 around 35% 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 AA.

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 on the next page). 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 5.8 and 6.5 years, respectively, compared to 5.1 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.

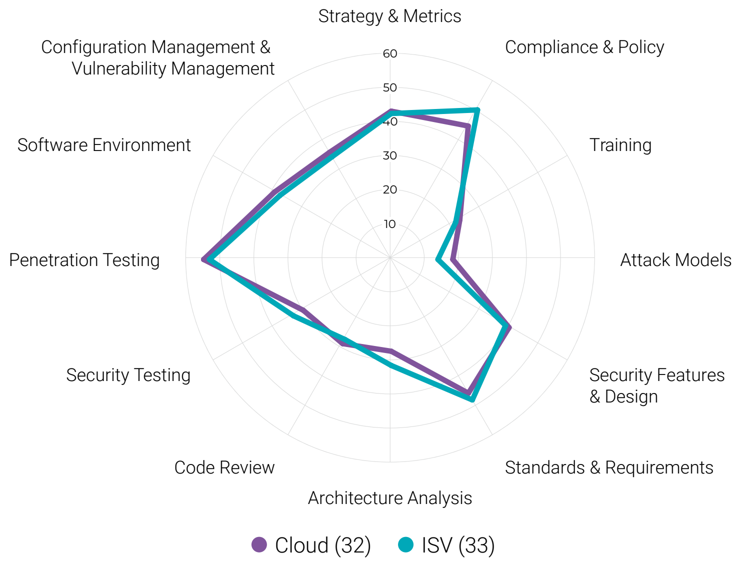


FIGURE 18. COMPARING CLOUD AND ISV VERTICALS. This diagram helps explain the differences, on a percentage scale, between practices in the Cloud and ISV verticals. Here, we see differences in the Compliance & Policy, Attack Models, and Architecture Analysis practices.

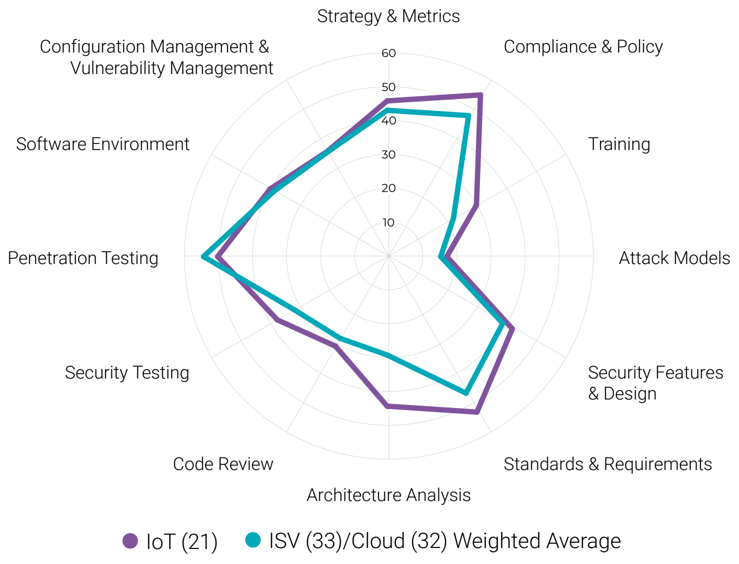


FIGURE 19. COMPARING IOT AND THE WEIGHTED AVERAGE OF ISV AND CLOUD. While the ISV and Cloud verticals are very similar, there are significant variations between IoT and those two verticals. The differences, on a percentage scale, in risk and deployment models, along with customer expectations, can explain 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. Technology firms have matched financial firms in Compliance & Policy, likely due to recent strengthening of 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 AA ([AA3.1 Have engineering teams lead AA process] and [AA3.3 Make the SSG available as an AA resource or mentor]) are more than two times higher among technology firms compared to financial firms, which 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.

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 AA 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.

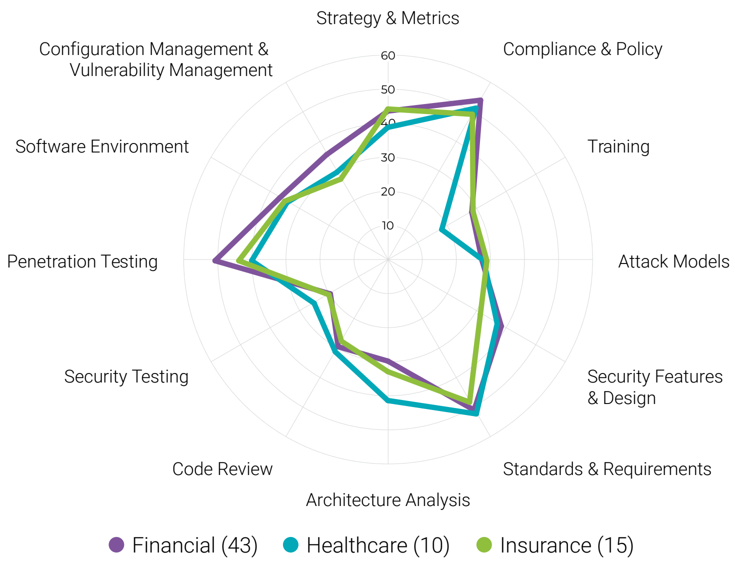


FIGURE 20. FINANCIAL VS. HEALTHCARE VS. INSURANCE. Even verticals that are similarly highly regulated exhibit significant differences in their SSIs. While they all have a focus on Compliance & Policy, there are significant differences, on a percentage scale, in most other practices, indicating that each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

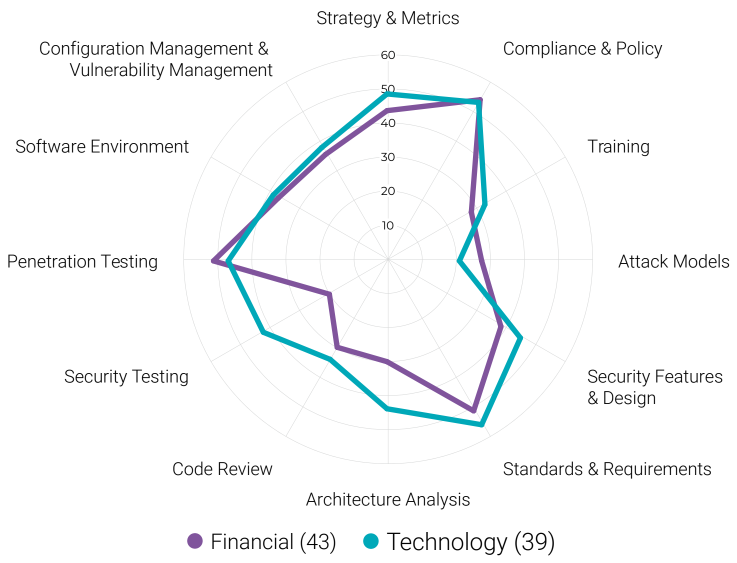


FIGURE 21. FINANCIAL VS. TECHNOLOGY. Technology firms appear to invest significantly more effort into in-depth design reviews, automation of security testing, and enablement of engineering teams to be self-sufficient, resulting in the differences, on a percentage scale, seen above. One potential explanation 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.

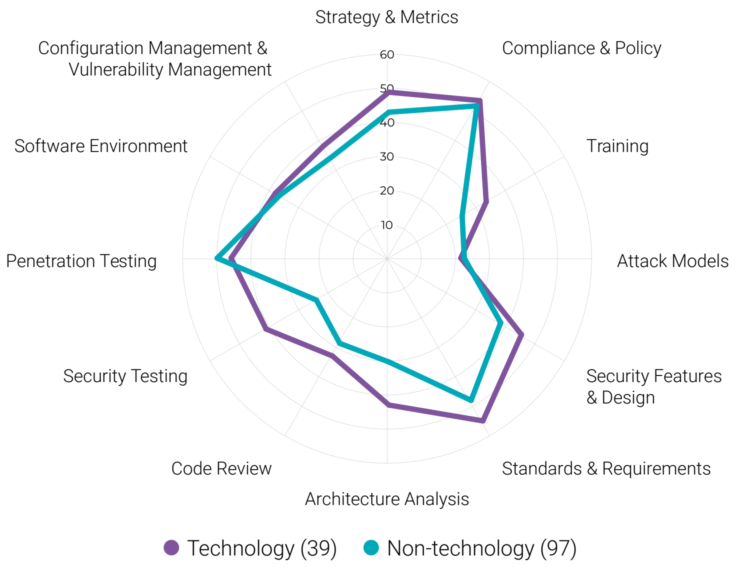


FIGURE 22. TECHNOLOGY VS. NON-TECHNOLOGY. Shown here is a comparison of the Technology vertical vs. the rest of the data pool on a percentage scale.

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 present the observation rate as a percentage (e.g., 78% 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 light blue) or at least 1.75 standard deviations below average (highlighted in gold). 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 gold color) and five activities with the largest differences between verticals (blue color). The activities in light orange appear to be uniformly applicable across all verticals, while those in dark blue 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 BSIMM14. 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 [AM2.9 Monitor automated asset creation] was introduced in BSIMM10. It has one of the largest differences between verticals, with its observation rate for the Financial vertical is significantly above the overall average. This is an indication that financial firms are early adopters of [AM2.9] 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).

GOVERNANCE								
ACTIVITY	CLOUD (OF 32)	FINANCIAL (OF 43)	FINTECH (OF 12)	HEALTHCARE (OF 10)	INSURANCE (OF 15)	IOT (OF 21)	ISV (OF 33)	TECH (OF 39)
STRATEGY & METRICS								
[SM1.1]	78%	70%	75%	90%	80%	100%	79%	97%
[SM1.3]	59%	63%	50%	60%	73%	48%	61%	62%
[SM1.4]	91%	95%	92%	90%	80%	95%	85%	92%
[SM2.1]	63%	65%	75%	50%	73%	38%	45%	59%
[SM2.2]	53%	58%	75%	20%	40%	52%	48%	64%
[SM2.3]	69%	37%	83%	60%	53%	62%	79%	54%
[SM2.6]	56%	63%	67%	30%	47%	62%	58%	64%
[SM2.7]	44%	44%	50%	70%	60%	43%	48%	54%
[SM3.1]	22%	23%	33%	0%	13%	33%	24%	36%
[SM3.2]	9%	14%	17%	10%	20%	33%	12%	26%
[SM3.3]	19%	33%	33%	20%	27%	29%	15%	26%
[SM3.4]	6%	7%	17%	10%	13%	5%	6%	3%
[SM3.5]	0%	0%	0%	0%	0%	0%	0%	0%
COMPLIANCE & POLICY								
[CP1.1]	69%	81%	83%	100%	80%	86%	82%	74%
[CP1.2]	84%	93%	100%	100%	100%	100%	85%	85%
[CP1.3]	66%	84%	67%	70%	80%	86%	73%	85%
[CP2.1]	44%	42%	58%	50%	33%	57%	45%	46%
[CP2.2]	41%	53%	33%	30%	40%	67%	42%	54%
[CP2.3]	50%	56%	67%	70%	47%	52%	55%	62%
[CP2.4]	41%	53%	42%	40%	53%	33%	52%	51%
[CP2.5]	59%	65%	67%	60%	47%	48%	70%	51%
[CP3.1]	13%	35%	50%	20%	40%	24%	18%	26%
[CP3.2]	19%	28%	8%	30%	20%	33%	27%	33%
[CP3.3]	16%	9%	8%	0%	7%	24%	12%	23%
TRAINING								
[T1.1]	56%	60%	58%	30%	53%	76%	48%	69%
[T1.7]	53%	53%	50%	40%	53%	52%	45%	59%
[T1.8]	31%	60%	42%	30%	60%	29%	33%	49%
[T2.5]	38%	19%	58%	30%	33%	38%	42%	44%
[T2.8]	19%	9%	0%	20%	13%	33%	24%	36%
[T2.9]	13%	30%	8%	20%	33%	43%	9%	38%
[T2.10]	16%	26%	25%	10%	27%	19%	18%	18%
[T2.11]	19%	26%	0%	30%	27%	33%	15%	28%
[T2.12]	22%	26%	25%	0%	20%	19%	21%	26%
[T3.1]	3%	9%	8%	0%	13%	0%	3%	10%
[T3.2]	6%	19%	8%	10%	13%	10%	0%	8%
[T3.6]	6%	5%	8%	0%	0%	14%	3%	13%

INTELLIGENCE								
ACTIVITY	CLOUD (OF 32)	FINANCIAL (OF 43)	FINTECH (OF 12)	HEALTHCARE (OF 10)	INSURANCE (OF 15)	IOT (OF 21)	ISV (OF 33)	TECH (OF 39)
ATTACK MODELS								
[AM1.2]	41%	79%	75%	100%	93%	19%	42%	38%
[AM1.3]	22%	44%	42%	60%	67%	33%	18%	33%
[AM1.5]	50%	77%	50%	90%	73%	67%	36%	64%
[AM2.1]	9%	16%	17%	10%	20%	10%	3%	10%
[AM2.6]	16%	9%	8%	0%	0%	5%	9%	23%
[AM2.7]	9%	16%	8%	10%	13%	5%	9%	10%
[AM2.8]	19%	16%	8%	10%	13%	24%	12%	18%
[AM2.9]	19%	19%	8%	0%	7%	14%	12%	3%
[AM3.2]	9%	7%	0%	10%	7%	5%	3%	3%
[AM3.4]	3%	12%	8%	10%	7%	10%	3%	15%
[AM3.5]	3%	9%	0%	10%	20%	5%	3%	15%
SECURITY FEATURES & DESIGN								
[SFD1.1]	75%	74%	83%	80%	67%	67%	73%	82%
[SFD1.2]	78%	70%	58%	80%	73%	86%	82%	77%
[SFD2.1]	34%	30%	58%	30%	7%	43%	33%	46%
[SFD2.2]	69%	44%	42%	40%	47%	71%	64%	64%
[SFD3.1]	9%	21%	8%	20%	13%	14%	9%	15%
[SFD3.2]	13%	19%	17%	10%	7%	10%	12%	21%
[SFD3.3]	3%	12%	8%	0%	7%	10%	0%	10%
STANDARDS & REQUIREMENTS								
[SR1.1]	59%	79%	67%	70%	80%	76%	64%	77%
[SR1.2]	84%	70%	67%	80%	67%	90%	85%	92%
[SR1.3]	63%	81%	83%	80%	67%	81%	76%	77%
[SR1.5]	75%	74%	100%	100%	73%	86%	85%	79%
[SR2.2]	50%	67%	42%	60%	80%	57%	42%	62%
[SR2.5]	41%	53%	50%	50%	47%	48%	45%	56%
[SR2.7]	47%	44%	92%	30%	33%	33%	45%	51%
[SR3.2]	0%	12%	8%	30%	20%	24%	12%	15%
[SR3.3]	22%	7%	25%	10%	0%	14%	9%	23%
[SR3.4]	16%	19%	8%	10%	13%	24%	15%	21%

SSDL TOUCHPOINTS								
ACTIVITY	CLOUD (OF 32)	FINANCIAL (OF 43)	FINTECH (OF 12)	HEALTHCARE (OF 10)	INSURANCE (OF 15)	IOT (OF 21)	ISV (OF 33)	TECH (OF 39)
ARCHITECTURE ANALYSIS								
[AA1.1]	91%	79%	92%	70%	87%	90%	91%	87%
[AA1.2]	34%	33%	33%	70%	40%	71%	45%	64%
[AA1.4]	28%	77%	67%	60%	73%	24%	24%	33%
[AA2.1]	22%	16%	8%	40%	20%	57%	27%	49%
[AA2.2]	19%	12%	8%	40%	13%	57%	24%	54%
[AA2.4]	22%	23%	33%	50%	27%	48%	33%	41%
[AA3.1]	13%	9%	8%	20%	20%	24%	18%	33%
[AA3.2]	3%	7%	0%	10%	7%	14%	3%	8%
[AA3.3]	13%	9%	8%	10%	7%	14%	15%	21%
CODE REVIEW								
[CR1.2]	66%	63%	67%	60%	60%	81%	61%	64%
[CR1.4]	81%	86%	92%	100%	87%	86%	88%	87%
[CR1.5]	47%	56%	75%	60%	47%	67%	55%	67%
[CR1.7]	47%	35%	58%	40%	40%	48%	48%	49%
[CR2.6]	28%	14%	33%	20%	13%	10%	24%	21%
[CR2.7]	16%	14%	17%	10%	20%	14%	3%	26%
[CR2.8]	16%	30%	17%	40%	20%	5%	15%	18%
[CR3.2]	9%	14%	8%	0%	7%	24%	3%	23%
[CR3.3]	3%	5%	17%	10%	7%	0%	0%	3%
[CR3.4]	0%	2%	0%	0%	0%	0%	0%	5%
[CR3.5]	3%	2%	0%	0%	0%	5%	3%	5%
SECURITY TESTING								
[ST1.1]	91%	70%	75%	100%	73%	95%	91%	97%
[ST1.3]	72%	51%	58%	70%	60%	86%	85%	87%
[ST1.4]	44%	37%	75%	60%	40%	62%	55%	64%
[ST2.4]	16%	9%	17%	0%	0%	29%	21%	33%
[ST2.5]	34%	16%	33%	10%	13%	29%	39%	38%
[ST2.6]	16%	7%	17%	0%	0%	43%	24%	44%
[ST3.3]	6%	0%	0%	10%	7%	33%	9%	36%
[ST3.4]	6%	0%	0%	0%	0%	0%	3%	10%
[ST3.5]	0%	0%	0%	0%	0%	0%	0%	8%
[ST3.6]	16%	5%	17%	0%	7%	0%	6%	0%

DEPLOYMENT								
ACTIVITY	CLOUD (OF 32)	FINANCIAL (OF 43)	FINTECH (OF 12)	HEALTHCARE (OF 10)	INSURANCE (OF 15)	IOT (OF 21)	ISV (OF 33)	TECH (OF 39)
PENETRATION TESTING								
[PT1.1]	97%	91%	100%	90%	93%	81%	97%	77%
[PT1.2]	84%	74%	100%	70%	67%	67%	91%	74%
[PT1.3]	63%	70%	75%	70%	67%	62%	70%	54%
[PT2.2]	34%	28%	58%	10%	13%	52%	36%	41%
[PT2.3]	63%	53%	58%	20%	47%	33%	55%	26%
[PT3.1]	31%	23%	50%	10%	7%	33%	18%	31%
[PT3.2]	16%	16%	42%	10%	13%	24%	9%	23%
SOFTWARE ENVIRONMENT								
[SE1.1]	66%	86%	67%	90%	80%	57%	61%	51%
[SE1.2]	88%	91%	100%	90%	100%	95%	76%	92%
[SE1.3]	81%	79%	83%	80%	93%	62%	82%	51%
[SE2.2]	50%	49%	58%	10%	40%	71%	52%	67%
[SE2.4]	38%	12%	25%	30%	7%	71%	42%	69%
[SE2.5]	59%	51%	83%	50%	53%	48%	52%	41%
[SE2.7]	56%	37%	58%	40%	33%	19%	48%	21%
[SE3.2]	6%	7%	8%	0%	0%	19%	9%	31%
[SE3.3]	16%	16%	25%	20%	13%	5%	18%	5%
[SE3.6]	13%	14%	17%	0%	0%	33%	12%	33%
[SE3.8]	6%	0%	8%	0%	0%	0%	3%	0%
[SE3.9]	0%	0%	0%	0%	0%	0%	0%	0%
CONFIGURATION MANAGEMENT & VULNERABILITY MANAGEMENT								
[CMVM1.1]	84%	93%	92%	90%	87%	86%	88%	92%
[CMVM1.2]	81%	67%	83%	70%	60%	81%	82%	77%
[CMVM1.3]	81%	65%	83%	80%	47%	81%	85%	87%
[CMVM2.1]	72%	72%	58%	70%	67%	71%	79%	67%
[CMVM2.3]	34%	49%	50%	40%	27%	29%	30%	38%
[CMVM3.1]	9%	12%	33%	0%	0%	10%	9%	18%
[CMVM3.2]	13%	21%	8%	0%	7%	24%	9%	31%
[CMVM3.3]	16%	21%	25%	20%	27%	5%	6%	13%
[CMVM3.4]	28%	23%	42%	10%	20%	14%	24%	18%
[CMVM3.5]	16%	21%	17%	10%	7%	14%	3%	10%
[CMVM3.6]	3%	2%	0%	0%	0%	5%	3%	5%
[CMVM3.7]	38%	21%	8%	0%	13%	48%	36%	41%
[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 BSIMM14. A light gold color in the Activity column shows the five activities with the least differences in observation rates between verticals, whereas a light blue color shows the five activities with the most differences. Blue and gold 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 participant SSIs have evolved over the past 15 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.) BSIMM14 describes the work of 11,000 SSG and satellite members (champions) working directly in software security, impacting the security efforts of almost 270,000 developers.

Forty-nine 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, 31 firms are on their second assessment, 10 firms are on their third, five are on their fourth, and two are on their fifth. 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
BSIMM14	130	81	31	10	5	304
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

Forty-nine of the 130 firms in BSIMM14 have been measured at least twice. On average, the time between first and second measurements for those 49 firms was 33.5 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 44 of the 49 firms and remained the same in two. Across all 49 firms, the score increased by an average of 13.1 (40.1%) 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 ([SR1.5 Identify open source])

Figure 25 shows the average normalized observation rate per practice for the 49 firms that have had a second assessment. Over the average of about 33.5 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 49 BSIMM14 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.

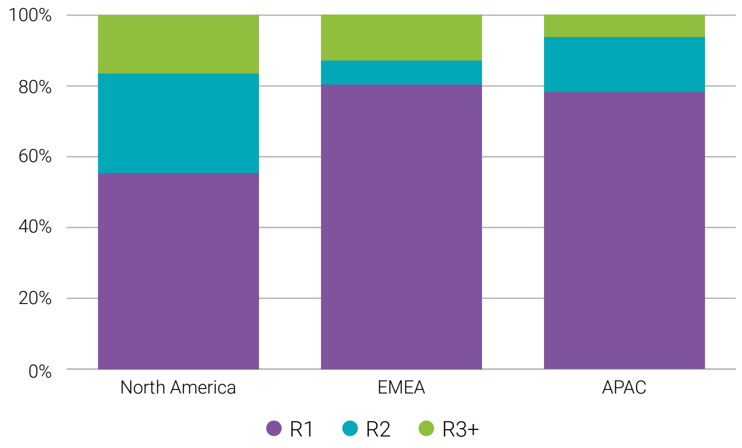


FIGURE 24. ONGOING USE OF THE BSIMM IN DRIVING ORGANIZATIONAL MATURITY. Organizations are continuing to do remeasurements to show that their efforts are achieving the desired results (e.g., about 56% of North America participants are on their first assessment).

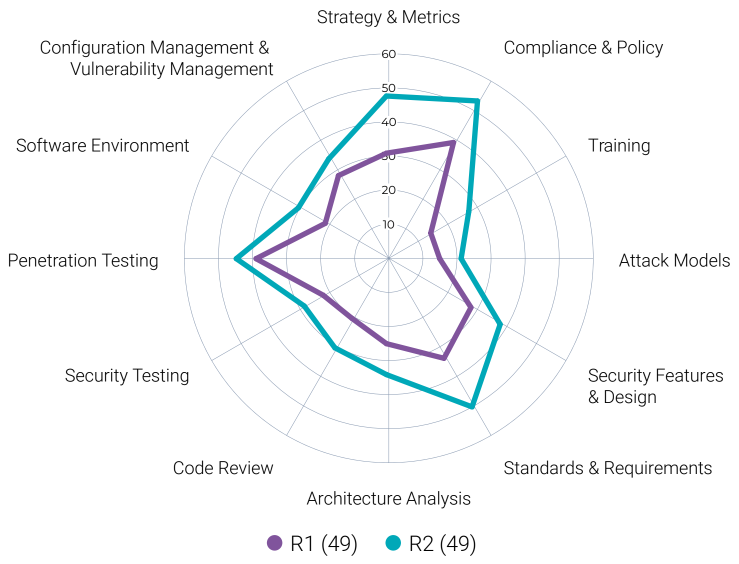


FIGURE 25. FIRMS ROUND 1 VS. FIRMS ROUND 2. This diagram illustrates the normalized observation rate change, on a percentage scale, in 49 firms between their first and second BSIMM assessments.

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

FIGURE 26. BSIMM14 REASSESSMENTS SCORECARD ROUND 1 VS. ROUND 2. This chart shows how 49 SSIs changed between their first and second assessments. Dark gold shows the top five activities with the most increase in observations by count. Light gold shows the next five activities with the most increase in observations by count.

CHANGES BETWEEN FIRST AND THIRD ASSESSMENTS

Eighteen of the 130 firms in BSIMM14 have been measured at least three times. On average, the time between first and third measurements for those 18 firms was 52.8 months. Although individual activities among the 12 practices come and go (as shown on the next page), in general, remeasurement over time shows a clear trend of increased maturity. The raw score went up in all 18 firms. Across all 18 firms, the score increased by an average of 21.9 (71.4%) 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 52.8 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.

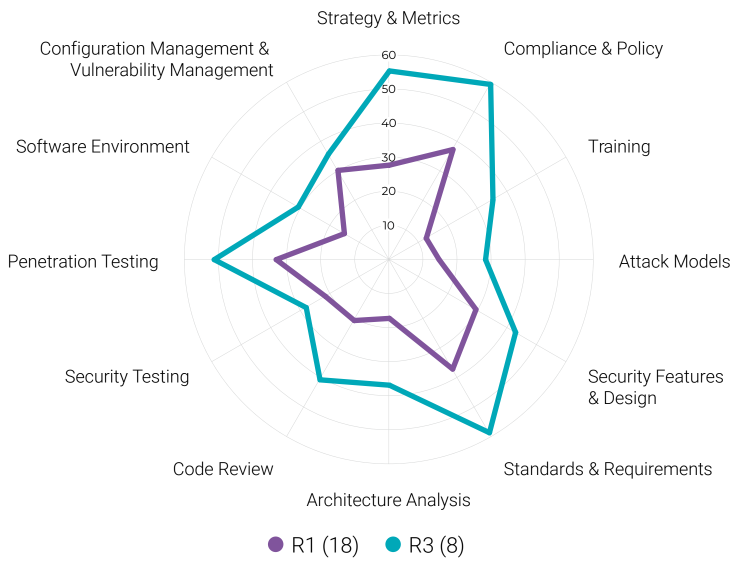


FIGURE 27. FIRMS ROUND 1 VS. FIRMS ROUND 3 SPIDER CHART. This diagram illustrates the normalized observation rate change, on a percentage scale, in 18 firms between their first and third BSIMM assessments.

GOVERNANCE		
ACTIVITY	BSIMM ROUND 1 (OF 18)	BSIMM ROUND 3 (OF 18)
STRATEGY & METRICS		
[SM1.1]	6	18
[SM1.3]	10	15
[SM1.4]	16	17
[SM2.1]	5	14
[SM2.2]	5	10
[SM2.3]	5	13
[SM2.6]	7	12
[SM2.7]	4	14
[SM3.1]	4	7
[SM3.2]	0	4
[SM3.3]	3	5
[SM3.4]	0	1
[SM3.5]	0	0
COMPLIANCE & POLICY		
[CP1.1]	11	18
[CP1.2]	14	17
[CP1.3]	7	13
[CP2.1]	6	10
[CP2.2]	5	9
[CP2.3]	7	13
[CP2.4]	5	11
[CP2.5]	9	13
[CP3.1]	4	9
[CP3.2]	5	4
[CP3.3]	1	1
TRAINING		
[T1.1]	11	12
[T1.7]	7	15
[T1.8]	2	13
[T2.5]	4	10
[T2.8]	2	5
[T2.9]	1	7
[T2.10]	0	1
[T2.11]	0	4
[T2.12]	0	4
[T3.1]	0	1
[T3.2]	0	4
[T3.6]	0	0

INTELLIGENCE		
ACTIVITY	BSIMM ROUND 1 (OF 18)	BSIMM ROUND 3 (OF 18)
ATTACK MODELS		
[AM1.2]	13	16
[AM1.3]	4	12
[AM1.5]	8	12
[AM2.1]	1	6
[AM2.6]	0	1
[AM2.7]	0	1
[AM2.8]	0	0
[AM2.9]	0	1
[AM3.2]	0	1
[AM3.4]	0	2
[AM3.5]	3	4
SECURITY FEATURES & DESIGN		
[SFD1.1]	15	16
[SFD1.2]	13	14
[SFD2.1]	2	5
[SFD2.2]	5	11
[SFD3.1]	0	2
[SFD3.2]	2	6
[SFD3.3]	0	0
STANDARDS & REQUIREMENTS		
[SR1.1]	12	15
[SR1.2]	12	17
[SR1.3]	14	17
[SR1.5]	4	13
[SR2.2]	6	13
[SR2.5]	4	10
[SR2.7]	3	8
[SR3.2]	4	5
[SR3.3]	2	3
[SR3.4]	6	5

SSDL TOUCHPOINTS		
ACTIVITY	BSIMM ROUND 1 (OF 18)	BSIMM ROUND 3 (OF 18)
ARCHITECTURE ANALYSIS		
[AA1.1]	16	18
[AA1.2]	1	7
[AA1.4]	8	13
[AA2.1]	1	7
[AA2.2]	0	6
[AA2.4]	1	5
[AA3.1]	1	3
[AA3.2]	0	0
[AA3.3]	0	1
CODE REVIEW		
[CR1.2]	13	16
[CR1.4]	11	18
[CR1.5]	3	10
[CR1.7]	3	15
[CR2.6]	1	5
[CR2.7]	4	5
[CR2.8]	6	8
[CR3.2]	0	2
[CR3.3]	0	2
[CR3.4]	0	0
[CR3.5]	0	0
SECURITY TESTING		
[ST1.1]	16	15
[ST1.3]	17	16
[ST1.4]	4	13
[ST2.4]	0	2
[ST2.5]	0	2
[ST2.6]	2	1
[ST3.3]	0	1
[ST3.4]	0	1
[ST3.5]	0	0
[ST3.6]	0	0

DEPLOYMENT		
ACTIVITY	BSIMM ROUND 1 (OF 18)	BSIMM ROUND 3 (OF 18)
PENETRATION TESTING		
[PT1.1]	16	18
[PT1.2]	10	16
[PT1.3]	10	14
[PT2.2]	1	7
[PT2.3]	4	5
[PT3.1]	1	2
[PT3.2]	0	3
SOFTWARE ENVIRONMENT		
[SE1.1]	8	15
[SE1.2]	15	17
[SE1.3]	0	10
[SE2.2]	3	4
[SE2.4]	2	3
[SE2.5]	1	9
[SE2.7]	0	4
[SE3.2]	2	2
[SE3.3]	2	2
[SE3.6]	0	1
[SE3.8]	0	0
[SE3.9]	0	0
CONFIG. MGMT. & VULN. MGMT.		
[CMVM1.1]	16	17
[CMVM1.2]	17	16
[CMVM1.3]	12	14
[CMVM2.1]	14	16
[CMVM2.3]	10	11
[CMVM3.1]	0	0
[CMVM3.2]	0	0
[CMVM3.3]	1	3
[CMVM3.4]	1	6
[CMVM3.5]	0	1
[CMVM3.6]	0	0
[CMVM3.7]	0	0
[CMVM3.8]	0	0

FIGURE 28. BSIMM14 REASSESSMENTS SCORECARD ROUND 1 VS. ROUND 3. This chart shows how 18 SSIs changed between their first and third assessments. Gold shows the top five activities with the most increase in observations by count. Light gold shows the next five activities with the most increase in observations by count.

G. DATA ANALYSIS: SATELLITE (SECURITY CHAMPIONS)

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 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 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 teal 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 3.8%). One way these firms are able to scale is by increasing the ratio of champions to developers, as shown by the teal bars (e.g., the bottom 20% have a satellite-to-developer ratio of 1.9%).

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 22.3 activities in the lowest scoring group (shown on the dark blue line), up to an average of 73.2 activities in the high scoring group (shown here as the top 20%).

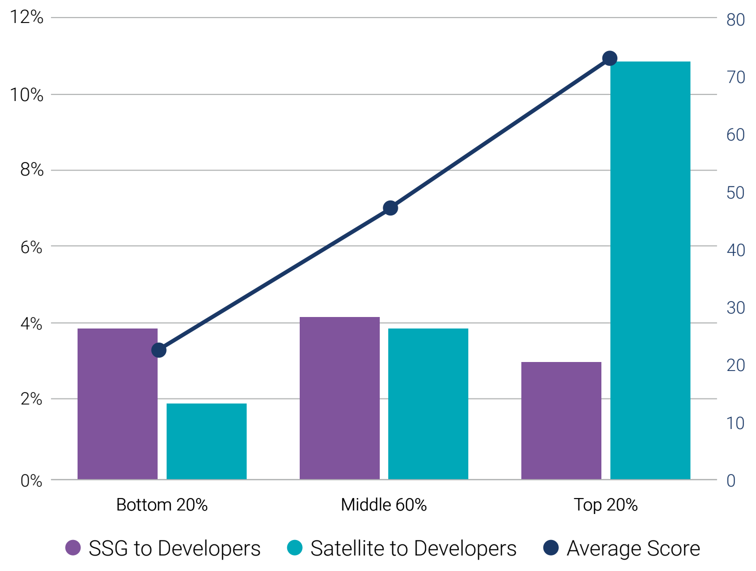


FIGURE 29. AVERAGE RATIO OF SSG AND SATELLITE SIZE TO DEVELOPERS FOR THREE SCORE BUCKETS. There is a strong correlation between security champions' support and overall BSIMM score (scale on the right).

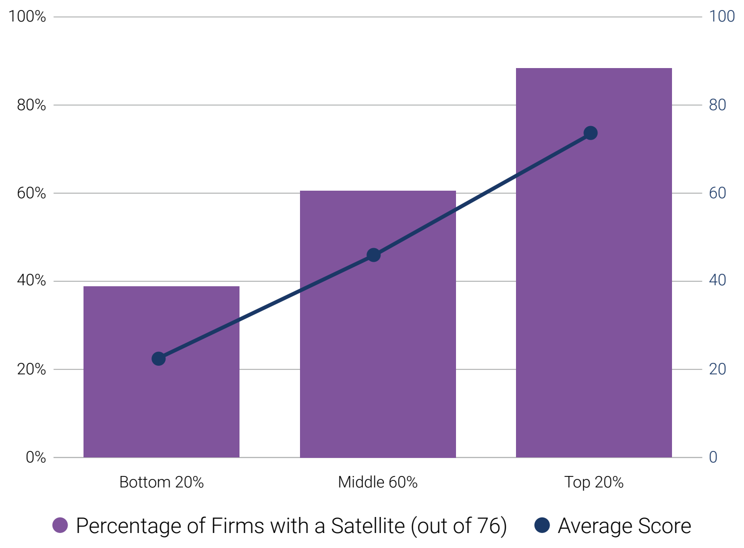


FIGURE 30. PERCENTAGE OF FIRMS THAT HAVE A SATELLITE, ORGANIZED IN THREE BUCKETS BY BSIMM SCORE. Presence of a satellite and average score (scale on the right) appear to be correlated, but we don't have enough data to say which is the cause and which is the effect. Here we see, for example, that in the bottom scoring 20% (about 15 firms) of the 76 (of 130) firms with a satellite, the average score was just over 20 compared to an average score of over 75 for the top scoring 20% with a satellite.

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, Training, and Standards & Requirements, the firms with a satellite also spend consistently more effort on defect discovery in the Architecture Analysis, Code Review, Security Testing, and Penetration 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 scores 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.

88% of firms in the highest scoring group have a champions program, compared to 38% in the lowest scoring group.

Seventy-eight percent of the 49 BSIMM14 firms that have been assessed more than once have a satellite, while 48% 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.

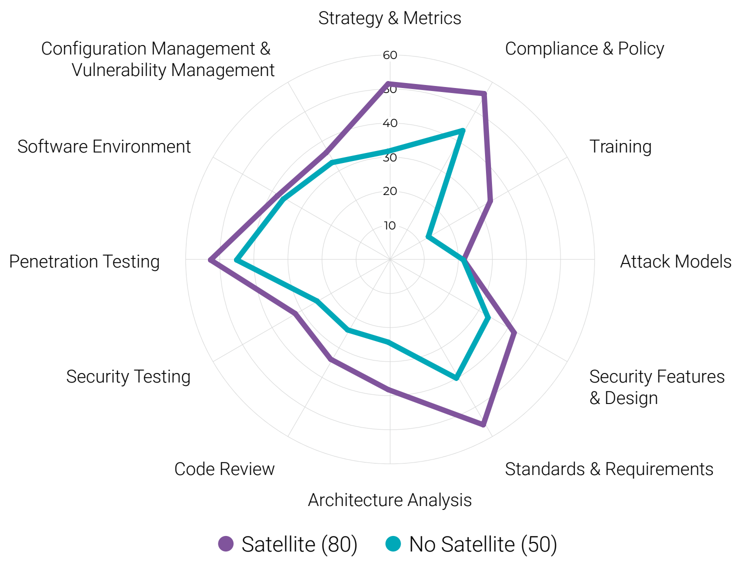


FIGURE 31. COMPARING FIRMS WITH AND WITHOUT A SATELLITE. The presence of a satellite (champions program) seems to correlate strongly with an increase in program maturity as evidenced by increased scores by practice on a percentage scale

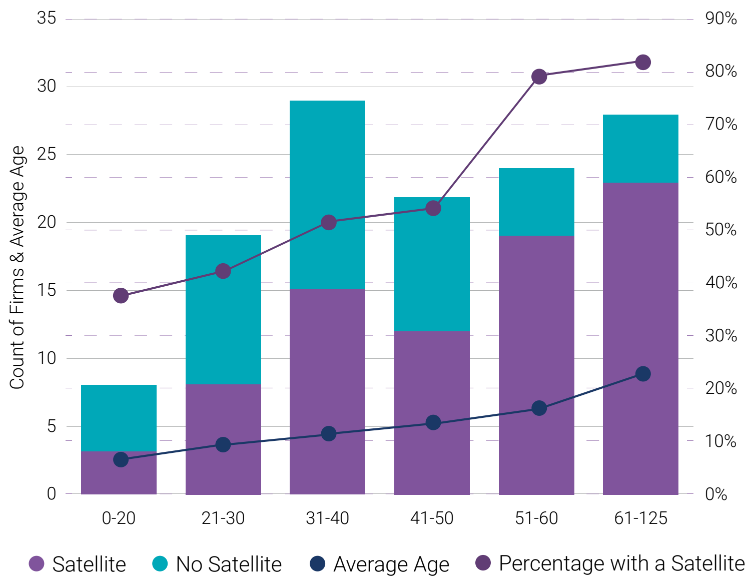


FIGURE 32. BSIMM SCORE DISTRIBUTION RELATIVE TO SATELLITE SIZE AND SSG AGE. Older SSIs (dark blue line) not only tend to have a higher BSIMM score (buckets 0-20, 21-30, etc.), they are also more likely to have a champions program (dark purple line).

H. DATA ANALYSIS: SSG

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.

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

SSG CHARACTERISTICS

As the BSIMM participants changed, 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 BSIMM7 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 data pool changed over time. In BSIMM14, 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 vs. 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 data pool, which causes the median satellite size to be 10 (50 of 130 firms had no satellite at the time of their current assessment); 57% of the 23 firms added for BSIMM14 had no satellite at assessment time, as seen in Figure 37.

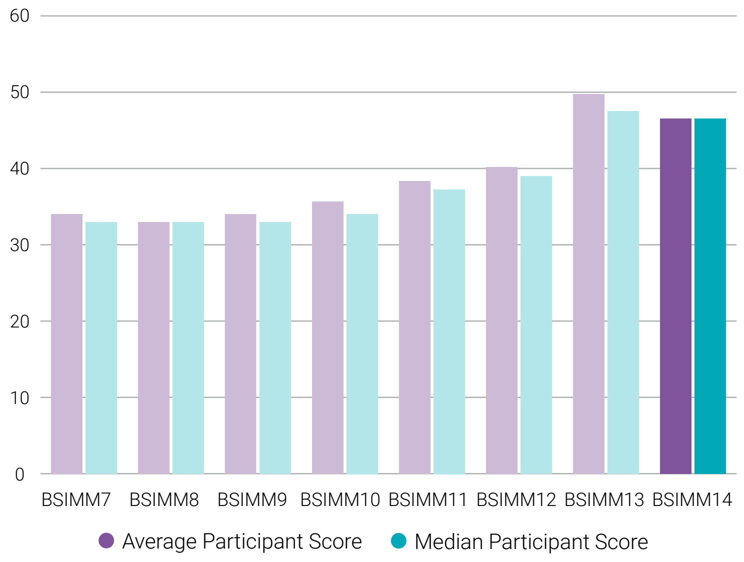


FIGURE 33. AVERAGE BSIMM PARTICIPANT SCORE. Adding firms with less experience decreased the average score from BSIMM7 through BSIMM8, even as remeasurements have shown that individual firm maturity increases over time.

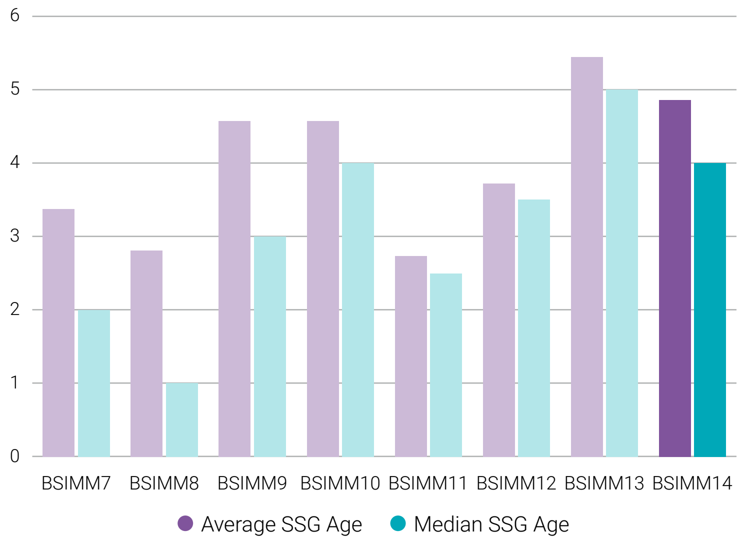


FIGURE 34. AVERAGE AND MEDIAN SSG AGE FOR NEW FIRMS ENTERING THE BSIMM DATA POOL. The median SSG age of firms entering BSIMM7 through BSIMM8 was declining and so did the average BSIMM score, while outliers in BSIMM7 and BSIMM8 resulted in a high average SSG age. Starting with BSIMM9, the median age of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

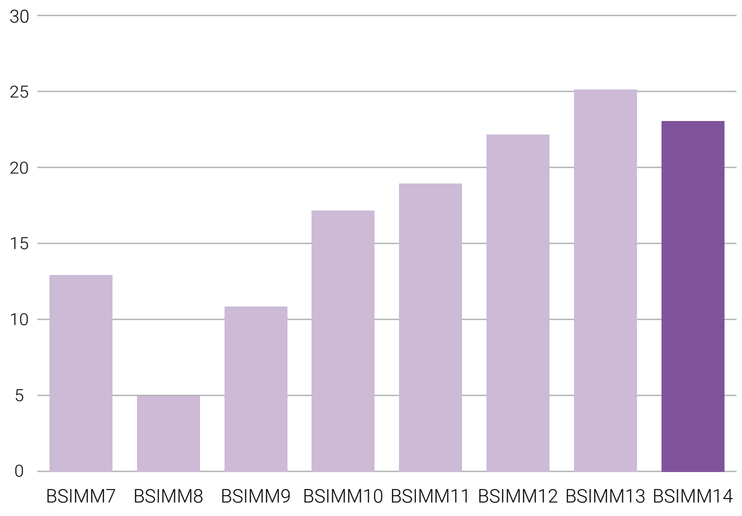


FIGURE 35. NUMBER OF FIRMS AGED OUT OF THE BSIMM DATA POOL. A total of 143 firms have aged out since BSIMM-V. Eighteen firms that had once aged out of the BSIMM data pool have subsequently rejoined with a new assessment.

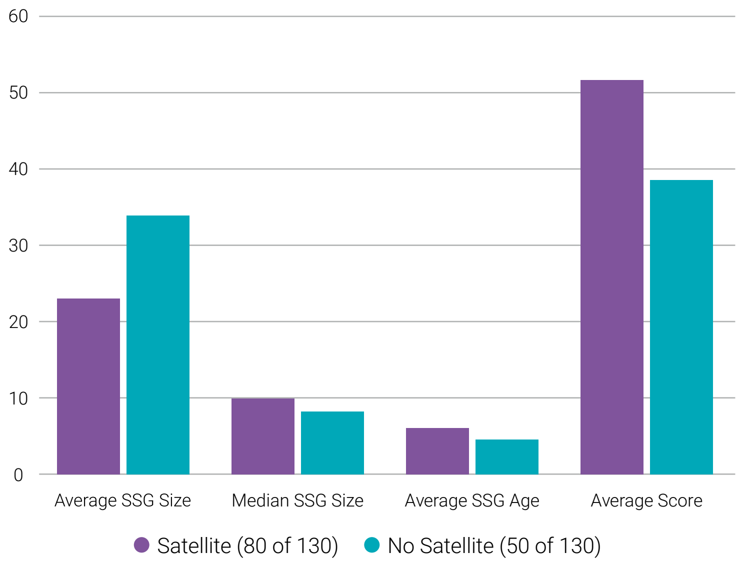


FIGURE 37. STATISTICS FOR FIRMS WITH AND WITHOUT A SATELLITE. This data appears to validate the notion that having more people, both centralized and distributed into engineering teams, helps SSIs achieve higher scores. For the 80 BSIMM14 firms with a satellite at last assessment time, the average satellite size was 93 with a median of 40 (not shown). We present the average and median SSG size to remove the impact of a few significant outliers.

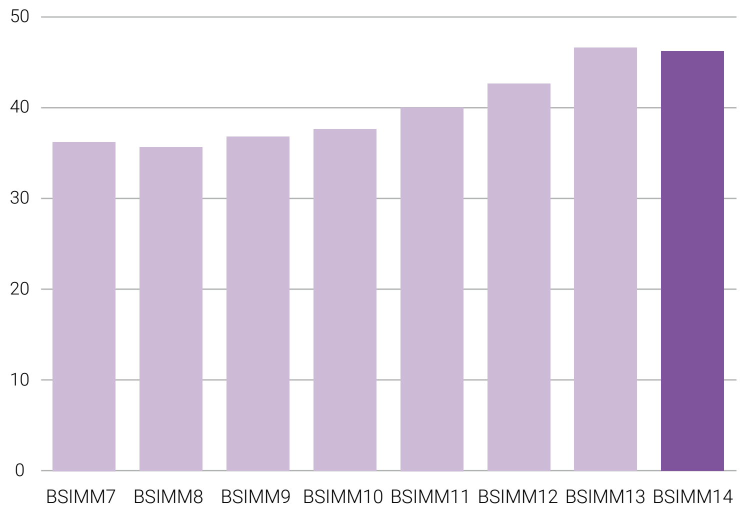


FIGURE 36. AVERAGE FINANCIAL VERTICAL FIRM SCORES. The average score across the Financial vertical followed the same pattern as the average score for AllFirms (shown in Figure 33). Even in such a mature vertical, we observe a rise in the average scores over time.

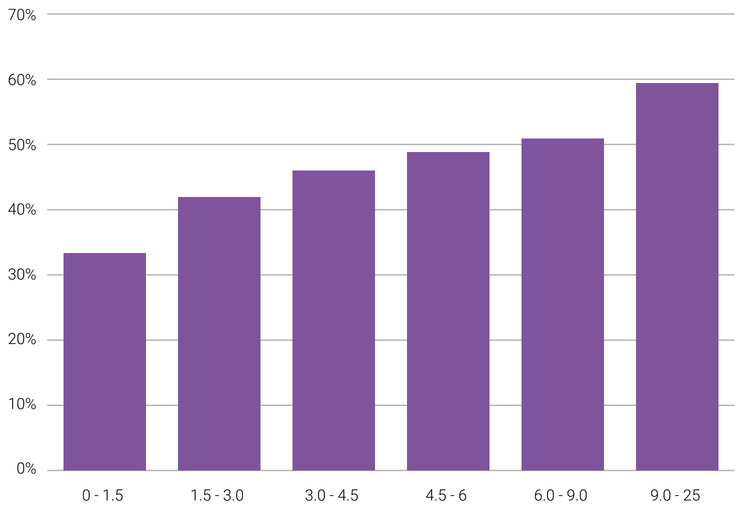


FIGURE 38. SSI SCORE DIVIDED BY AGE. By notionally organizing SSIs into emerging, maturing, and enabling phases by age in years, we see a steady growth in score as SSIs mature.

SSG CHANGES BASED ON AGE

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 BSIMM14 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 purple 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 (teal 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, Security Features & Design, 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 (purple line) and the beginning of their maturing phase (teal 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 penetration 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.

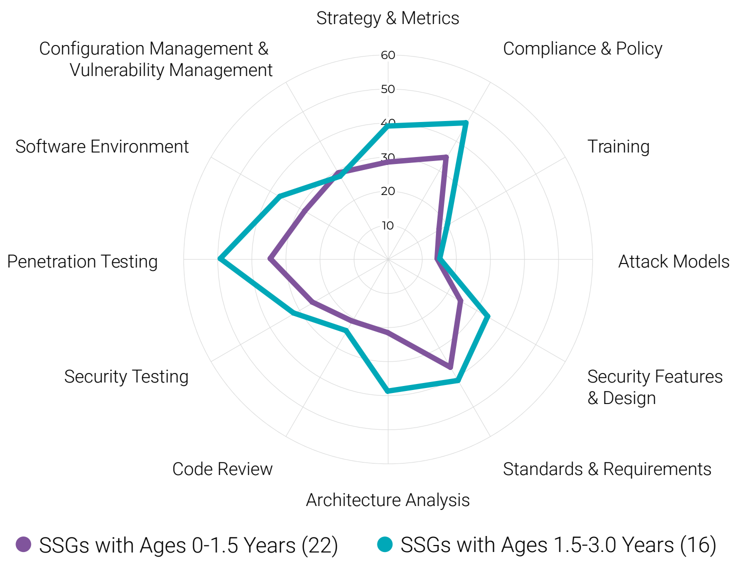


FIGURE 39. COMPARING EMERGING SSIs. As emerging SSIs move from initial discovery steps (purple line) toward defining and rolling out the program (teal line), they invest in Strategy & Metrics, Compliance & Policy, Standards & Requirements, Penetration Testing, and Architecture Analysis. This tracks with recommendations in Appendix B on how to start an SSI, where almost 45% of all recommended activities in Figure 12 are from these four practices.

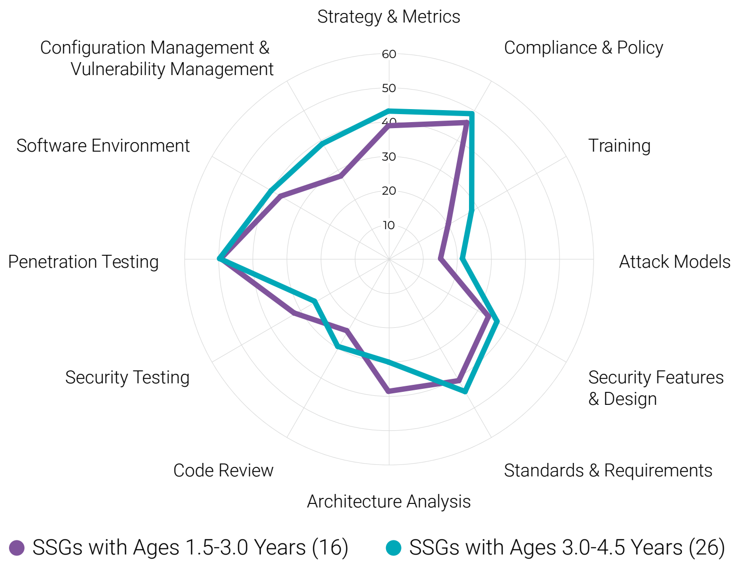


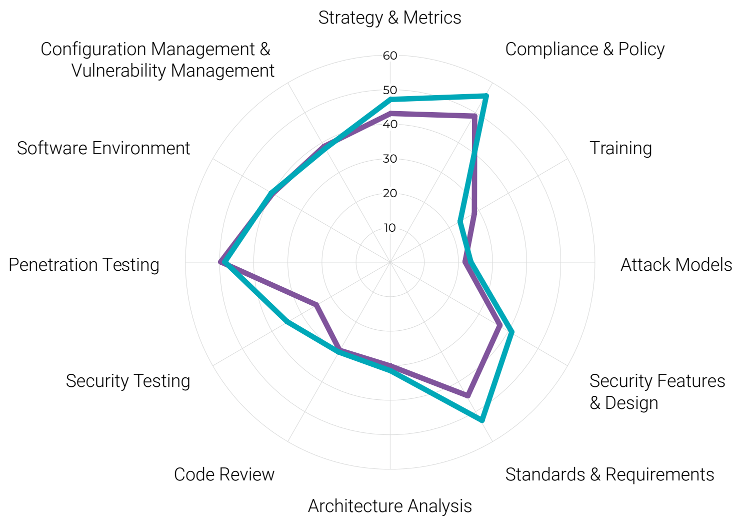
FIGURE 40. COMPARING LATE EMERGING TO EARLY MATURING SSIs. As firms move from emerging to maturing, the average score increase is relatively small. This aligns with our qualitative observations in Appendix B that these firms often focus more on the scale, efficiency, and effectiveness of existing activities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

Some factors specific to verticals might significantly influence the overall shape of the spiders. For example, 20% of firms with an SSG age between six and nine years are in the Insurance vertical as compared to 6.7% in the entire BSIMM14 data pool. Similarly, 27% of the firms with an SSG age above nine years are in the Financial vertical versus 19% 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.

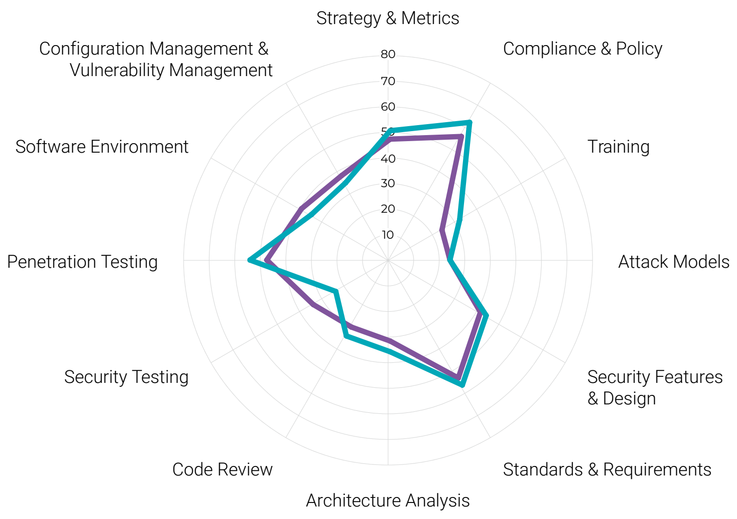
One potential explanation for the dip in Security Testing shown in Figure 42 is that the Financial vertical has one of the lowest observation rates for this practice. For the spike in the Penetration Testing practice, almost 60% of all firms in the age bucket between six and nine years are either in 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 Attack Models, Security Features & Design, Standards & Requirements, 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.



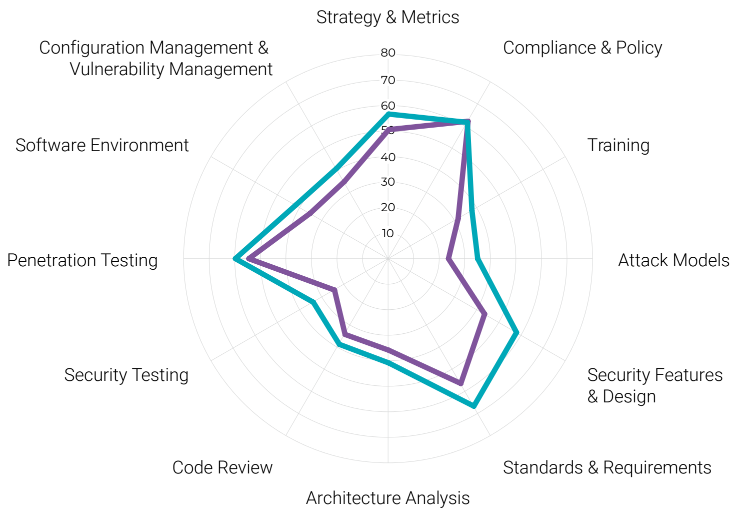
● SSGs with Ages 3.0-4.5 Years (26) ● SSGs with Ages 4.5-6.0 Years (21)

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 Security Testing practice and decreased investment in the Penetration Testing practice.



● SSGs with Ages 4.5-6.0 Years (21) ● SSGs with Ages 6.0-9.0 Years (26)

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. 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.



● SSGs with Ages 6.0-9.0 Years (26) ● SSGs with Ages over 9.0 Years (19)

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.

BSIMM14