BSIMM 11

BSIMM11 AUTHORS

Created by Sammy Migues, John Steven, and Mike Ware

Sammy Migues
Principal Scientist at Synopsys
Sammy works primarily with senior managers on entrepreneurial innovation, practical business solutions, and business process optimization. At Synopsys, he started the internal knowledge management program as well as commercial offerings for CBT and ILT software security training, smart grid security, and the management consulting practice for software security initiative creation and improvement, metrics, and DevSecOps transformation. Migues is also a creator and the maintainer of the BSIMM, the only study of its kind to capture the actual software security practices in nearly 200 firms around the world.

John Steven
Founding Principal of Aedify Security
John provides strategic consulting to software security initiatives and serves as technical advisor to application security and cloud security vendors. John is also CTO of ZeroNorth. For two decades, John led technical direction at Cigital. He founded Cigital spin-off Codiscope as its CTO in 2015. Both firms were acquired by Synopsys in 2016, where John served as Senior Director of Security Technology and Applied Research. A BSIMM author and trusted advisor to security executives, John is keenly interested in enabling software-defined security governance, using his unparalleled experience with a broad range of security tools andtechniques to build mature security initiatives.

Mike Ware
Sr. Director of Technology at Synopsys
Mike leads the consulting technology practices for Synopsys within the Software Integrity Group (SIG), based in the Eastern Panhandle of WV. Having undertaken leadership roles in both client management and technology thought leadership since 2008, Mike has led the implementation of our client's largest software security initiatives. A BSIMM co-author, Mike focuses on relentlessly understanding the business and technology drivers for software security as an industry and innovating SIG's portfolio of software security capabilities and solutions.

BSIMM LICENSE

EXECUTIVE SUMMARY

The BSIMM is the result of a multiyear study of real-world software security initiatives (SSIs). We present BSIMM11 as built directly out of data observed in 130 firms. These firms are listed in the Acknowledgments section.

The BSIMM is primarily a measuring stick for software security. The best way to use it is to compare and contrast your own initiative with the data about what other organizations are doing. The BSIMM also functions as a roadmap for an SSI. You can identify your own goals and objectives, then refer to the BSIMM to determine which additional activities make sense for you.

The purpose of the BSIMM is to quantify the activities carried out by various kinds of SSIs across many organizations. Because these initiatives use different methodologies and different terminology, the BSIMM requires a framework that allows us to describe any initiative in a uniform way. Our software security framework (SSF) and activity descriptions provide a common vocabulary for explaining the salient elements of an SSI, thereby allowing us to compare initiatives that use different terms, operate at different scales, exist in different parts of the organizational chart, operate in different vertical markets, or create different work products.

We understand that not all organizations need to achieve the same security goals, but we believe all organizations can benefit from using a common measuring stick. The BSIMM is not a traditional maturity model where a set of activities are repeated at multiple levels of depth and breadth—do something at level 1, do it more at level 2, do it better at level 3, and so on. Instead, the BSIMM comprises a set of unique activities, with activity levels used only to distinguish the relative frequency with which the activities are observed in organizations. Frequently observed activities are designated as level 1, less frequently observed activities are designated level 2, and infrequently observed activities are designated level 3. Table A shows the natural groupings of activities.

In the rapidly changing software security field, understanding what most, some, and few other organizations are doing in their SSIs can directly inform your own strategy.

We hold the scorecards for individual firms in confidence, but we publish aggregate data describing the number of times we have observed each activity (see the BSIMM11 Scorecard in Part Two). We also publish observations about subsets (such as industry verticals) when our sample size for the subset is large enough to guarantee anonymity.

NATURAL GROUPINGS OF ACTIVITIES

LEVEL 1	SOFTWARE ENVIRONMENT (SE)	OBSERVATIONS
Activities in this practice observed most frequently	[SE1.1] Use application input monitoring.	73
	[SE1.2] Ensure host and network security basics are in place.	121
LEVEL 2	**SOFTWARE ENVIRONMENT (SE)**	**OBSERVATIONS**
Activities in this practice observed relatively frequently	[SE2.2] Define secure deployment parameters and configurations.	39
	[SE2.4] Protect code integrity.	33
	[SE2.5] Use application containers.	31
	[SE2.6] Ensure cloud security basics.	36
LEVEL 3	**SOFTWARE ENVIRONMENT (SE)**	**OBSERVATIONS**
Activities in this practice observed least frequently or are newly added	[SE3.2] Use code protection.	13
	[SE3.3] Use application behavior monitoring and diagnostics.	7
	[SE3.5] Use orchestration for containers and virtualized environments.	22
	[SE3.6] Enhance application inventory with operations bill of materials.	12

Table A. Natural Groupings of Activities. The observation rate leads to natural groupings of activities into those seen most often, often, and least often.

KEY TAKEAWAYS FOR BSIMM11

This annual BSIMM report presents observed quantitative data on actual software security practices across 130 organizations. You will see these numbers throughout the document relative to software security activities, vertical markets, champions programs, reporting lines, change over time, and much more. Each year, we gather these data through hundreds of interviews with the individuals directly involved in making software security happen in their organizations.

During these interviews, we also learn what organizations think regarding how their software security initiatives (SSIs) operate today and how they might need to operate in the future. We express that qualitative data throughout this document as, for example, a guide to starting and maturing an initiative, as themes across many initiatives, and as real-world examples within activity descriptions.

Below, we summarize some of that data as BSIMM11's key takeaways.

1. **Engineering-led software security efforts are having success contributing to DevOps value streams in pursuit of resiliency.**
   - CI/CD instrumentation and operations orchestration have become a normal part of the approach to software security for some organizations and are influencing how SSIs are organized, designed, and executed. As an example, increasingly, we see SSIs report through technology groups (20 of 130 firms) or to a CTO (21 of 130 firms) up to the CEO.
   - Engineering-led firms are changing how software security groups find internal talent and use it to scale their depth and breadth. For instance, the satellite role (e.g., champion) is evolving rapidly in firms embracing DevOps and DevSecOps to recruit members from cloud and related security-oriented roles and increasingly have them apply their expertise as code for everyone's benefit.
   - Some centralized software security groups (SSGs) are building bridges with the talent aggregating within cloud security and digital transformation teams, as well as within forward-looking development teams. This might reflect a shift to SSG members delivering secure application functionality in places along the value stream rather than simply putting in place governance as defect discovery.
   - Cloud service providers offer a shared responsibility model that makes SSIs contend with what is effectively compulsory outsourcing of at least parts of security architecture, feature provisioning, and other SSI practice areas that are traditionally done 100% locally. Ensuring proper risk management across all cloud provider resources for all engineering teams is a key concern.
2. **Software-defined security governance is no longer just aspirational.**
   - Digital transformation initiatives are incorporating security activities into their efforts to provide some manner of CI/CD-pipeline-as-a-service and more broadly make software delivery self-service for development teams. In so doing, previously high-friction security activities conducted by the SSG out-of-band and at gates are becoming more transparent and are increasingly applied by default as part of pipeline execution. Some have made significant progress on a goal of governance-as-code.
   - The three activities added for BSIMM10 described a clear arc—software-defined lifecycle governance, software-assisted monitoring of software-defined asset creation, and automated verification of software-defined infrastructure—which shows that some organizations are actively working on ways to speed up security to match the speed with which the business delivers functionality to market. BSIMM11 captures the continued evolution of this arc with the addition of activities for implementing event-driven security testing [ST3.6] and publishing risk data for deployable artifacts [CMVM3.6].
   - Assigning repetitive analysis and procedural tasks to bots, sensors, and other automation is increasingly the way organizations choose to address people and skills shortages, as well as cadence management problems. For example, converting to algorithms the decision-making on what security tools to run when often removes a variety of human processes from the application lifecycle.
3. **Security is becoming part of a quality practice, which is being recognized as part of reliability, all in pursuit of resilience.**
   - Many activities that were traditionally siloed and expert-driven are evolving into a set of technologies and processes that provide needed insight to a target audience and do so in cadence. As a technology example, there are now SAST tools ranging from fast-running open source that target specific single-language issues to full-program behavioral analysis tools with much longer execution cycles. While results from the former often support the quality and reliability goals in development, it is usually the latter that are able to support the security and resilience goals in the governance cycles.

- Engineering groups are building capabilities focused on inventorying software, cataloging its components, and capturing the means by which it was built, configured, and deployed. Rather than attempting to find all problems before deployment, for example, many DevOps groups continue to optimize by knowing what is running where and by creating capabilities to restart or re-deploy patched or misbehaving software, which focuses on groundwork for resilience in response to a variety of drivers—security telemetry included.
- Because they're often free, easy to find, and easy to use, engineering-led cultures tend to start with open source or "freemium" security tools to accomplish various in-band automated code review and testing activities. The organization's larger, commercial security testing tools that execute over extended periods typically continue to get used at checkpoints relevant to governance groups, with those tests usually being out-of-band of CI/CD pipelines due to the time taken to execute and the friction introduced.
- Rapidly evolving cloud and platform security features, as well as branching strategies, deployment strategies, site reliability approaches, and so on, are making it easier for some engineering teams to align more closely with resilience as a primary goal for their software rather than simply security or quality.
- In the software world, many organizations now have engineers for security, quality, reliability, and resilience (in addition to the traditional core, infrastructure, network, and other types of engineers). These roles each have their own objectives, timelines, cultures, and guardrails, as well as their own group of people telling them what they've done wrong without necessarily collaborating broadly on what works best for everyone. No one can boil the ocean and manage all this change at once, necessitating bridge-building and prioritization in pursuit of organizational success over group goals.

4. **"Shift Left" is becoming "Shift Everywhere."**
    - Although shift left has been promoted as doing some security testing during development, that is a large simplification of what we meant. More accurately today, some secure software development lifecycles (SSDLs) seek to conduct an activity as quickly as possible with the highest fidelity as soon as the artifacts on which that activity depend are made available. Sometimes, that's to the left of where you're doing things today, but often times, it's to the right. In addition, technology trends naturally require shifting right to produce rapid and accurate telemetry from modern languages, frameworks, and software orchestration.
    - Established practices such as secure code review are leveraging enhanced source code management features to allow review during multiple phases. For example, shift left to initial code commits and shift right to augment metadata offered as part of pull requests sent to repository maintainers when code is finished and tested. These options reflect a desire to present results both where they can be achieved the soonest and where they will be most impactful.
    - Some organizations evaluating defect discovery tools and services are showing a growing preference for continuous event-based security telemetry throughout a value stream rather than a single point-in-time analysis.
    - Those organizations attempting to maintain accurate software inventory data are discovering the need to align efforts across source code content management, the build process, the deployment process, and the operations environment, where inventory granularity and content will likely be different with each view and will also change frequently. Such organizations are struggling to maintain the effectiveness of their existing inventory efforts while also adapting to new software lifecycles, software architecture changes and any underlying software, deployment, and cloud technologies changes happening in response to the engineering self-service trends and the digital transformation sea change.

Of course, other large trends demand attention and perhaps haven't yet shown their true impact on software security. In particular, and anecdotally, the current world and political climate has caused significant changes in software security processes, technology, and resourcing. As one point, we hear stories of a significant slow-down in hiring in many organizations and a mandate to get both this year's and next year's goals done with existing staff and technology. Primarily, we see this implemented as a significant acceleration in process automation, in applying some manner of intelligence through sensors to prevent people from becoming process blockers, and in the start of a cultural acceptance that going faster means not everything (e.g., not all desired security testing) can be done in-band of the delivery lifecycle before deployment.

ACKNOWLEDGMENTS

Our thanks to the 130 executives from the SSIs we studied from around the world to create BSIMM11, including those who choose to remain anonymous.

Adobe
Aetna
Alibaba
Ally Bank
Autodesk
Axway
Bank of America
Bell
BMO Financial Group
Black Duck Software
Black Knight Financial Services
Box
Canadian Imperial Bank of Commerce
City National Bank
Cisco
Citigroup
Dahua
Depository Trust & Clearing Corporation
Eli Lilly
Equifax
Experian
F-Secure
Fannie Mae
Freddie Mac
General Electric
Genetec
Global Payments
HCA Healthcare
Highmark Health Solutions
Honeywell
Horizon Healthcare Services, Inc.
HSBC
iPipeline
Johnson & Johnson
JPMorgan Chase & Co.
Lenovo
MassMutual
McKesson
Medtronic
Morningstar
Navient
Navy Federal Credit Union
NCR
NEC Platforms
NetApp
NewsCorp
NVIDIA
PayPal
Pegasystems
Principal Financial Group
Royal Bank of Canada
SambaSafety
ServiceNow
Synopsys
TD Ameritrade
The Home Depot
The Vanguard Group
Trainline
Trane
U.S. Bank
Veritas
Verizon
Verizon Media
Wells Fargo
Zendesk

Our thanks also to the nearly 120 individuals who helped gather the data for the BSIMM.

In particular, we thank Matthew Chartrand, Sagar Dongre, Michael Doyle, Eli Erlikhman, Jacob Ewers, Stephen Gardner, Iman Louis, Daniel Lyon, Alistair Nash, Kevin Nassery, Donald Pollicino, and Denis Sheridan. In addition, we give a special thank you to Kathy Clark-Fisher, whose behind-the-scenes work keeps the BSIMM science project, conferences, and community on track.

Data for the BSIMM were captured by Synopsys. BSIMM10 and BSIMM11 were authored by Sammy Migues, Mike Ware, and John Steven. BSIMM4–BSIMM9 were authored by Gary McGraw, Ph.D., Sammy Migues, and Jacob West. BSIMM1–BSIMM3 were authored by Gary McGraw, Ph.D., Brian Chess, Ph.D., and Sammy Migues.

BSIMM11 TABLE OF CONTENTS

APPENDIX

BSIMM11 LIST OF TABLES

BSIMM11 LIST OF FIGURES

PART ONE: BACKGROUND

The BSIMM is now in its eleventh iteration. In this section, we talk about its history and the underlying model, as well as the changes we made for BSIMM11. We describe the roles we typically see in an SSI and some related terminology. We conclude this section with guidance on how to use the BSIMM to start, mature, and measure your own SSI.

BSIMM HISTORY

We built the first version of the BSIMM a little over a decade ago (Fall 2008) as follows:

- **We relied on our own knowledge of software security practices to create the SSF (found in Part Two).**
- **We conducted a series of in-person interviews with nine executives in charge of SSIs. From these interviews, we identified a set of common activities, which we organized according to the SSF.**
- **We then created scorecards for each of the nine initiatives that showed which activities the initiatives carry out. To validate our work, we asked each participating firm to review the framework, the practices, and the scorecard we created for their initiative.**

Today, we continue to evolve the model by looking for new activities as participants are added and as current participants are remeasured. We also adjust the model according to observation rates for each of the activities.

THE MODEL

The BSIMM is a data-driven model that evolves over time. We have added, deleted, and adjusted the levels of various activities based on the data observed as the project has evolved. To preserve backwards compatibility, we make all changes by adding new activity labels to the model, even when an activity has simply changed levels (e.g., we add a new CRx.x label for both new and moved activities in the Code Review practice). 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. When considering whether to move an activity between levels, we use the results of an intralevel standard deviation analysis and the trend in observation counts.

Whenever possible, we use an in-person interview technique to conduct BSIMM assessments, done with a total of 211 firms so far. In addition, we conducted assessments for 10 organizations that have rejoined the community after once aging out. In 50 cases, we assessed the software security group (SSG) and one or more business units as part of creating the corporate SSI view.

For most organizations, we create a single aggregated scorecard, whereas in others, we create individual scorecards for the SSG and each business unit assessed. However, each firm is represented by only one set of data in the model published here. ("Table 7. BSIMM Numbers Over Time" in the appendix shows changes in the data pool over time.)

As a descriptive model, the only goal of the BSIMM is to observe and report. We like to say we visited a neighborhood to see what was happening and observed that "there are robot vacuum cleaners in X of the Y houses we visited." Note that the BSIMM does not say, "all houses must have robot vacuum cleaners," "robots are the only acceptable kind of vacuum cleaners," "vacuum cleaners must be used every day," or any other value judgements. We offer simple observations simply reported.

Of course, during our assessment efforts across hundreds of organizations, we also make qualitative observations about how SSIs are evolving and report many of those as key takeaways, themes, and other topical discussions in this document.

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 modeling SSIs has either described the experience of a single organization or offered prescriptive guidance based purely on a combination of personal experience and opinion.

“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.”

CREATING BSIMM11 FROM BSIMM10

BSIMM11 includes updated activity descriptions, data from 130 firms in multiple vertical markets, and a longitudinal study. For BSIMM11, we added 27 firms and removed 19, resulting in a data pool of 130 firms. In addition, 14 firms conducted reassessments to update their scorecards, and we assessed additional business units for five firms.

We used the resulting observation counts to refine activity placement in the framework, which resulted in moving four activities to different levels. In addition, we added two newly observed activities, resulting in a total of 121 activities in BSIMM11.

Changes made for BSIMM11

- [T2.6 Include security resources in onboarding] became T1.8.
- [CR2.5 Assign tool mentors] became CR1.7.
- [SE3.4 Use application containers] 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.

We also carefully considered but did not adjust [CMVM2.2 Track software bugs found in operations through the fix process] at this time; we will do so if the observation rate continues to increase. Similarly, we considered and did not adjust [ST2.1 Integrate black-box security tools into the QA process] but will do so if the observation rate continues to increase.

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 budgets] became SM2.5 in BSIMM3 and is now SM3.3 due to its observation rate remaining fairly static while other activities became observed much more frequently. Similarly, the BSIMM1 activity [SR1.4 Use coding standards] became SR2.6 in BSIMM6 and is now SR3.3 as its observation rate has decreased. To date, no activity has migrated from level 3 to level 1, but we see significant observation increases in recently added cloud- and DevOps-related activities.

NEW ACTIVITIES					
ACTIVITY	BSIMM7 OBSERVATIONS	BSIMM8 OBSERVATIONS	BSIMM9 OBSERVATIONS	BSIMM10 OBSERVATIONS	BSIMM11 OBSERVATIONS
SE3.4 (now SE2.5)	0	4	11	14	31
SE3.5			0	5	22
SE3.6			0	3	12
SE3.7 (now SE2.6)			0	9	36
SM3.4				0	1
AM3.3				0	4
CMVM3.5				0	8
ST3.6					0
CMVM3.6					0

Table 1. New Activities. Some of the most recently added activities have seen exceptional growth in observation counts, perhaps demonstrating their widespread utility.

We noted in BSIMM7 that, for the first time, an activity ([AA3.2 Drive analysis results into standard architecture patterns]) was not observed in the current dataset, and there were no new observations of AA3.2 for BSIMM8. AA3.2 did have two observations in BSIMM9 and one observation in both BSIMM10 and BSIMM11; there are currently no activities with zero observations (except for the two just added).

Where Do Old Activities Go?

We continue to ponder the question, "Where do activities go when no one does them anymore?" In addition to SR3.3 mentioned above, we've noticed that the observation rate for other seemingly useful activities has decreased significantly in recent years:

- [T3.6 Identify new satellite members through observation] – observed in 11 of 51 firms in BSIMM4 and 1 of 130 firms in BSIMM11.
- [AA3.3 Make the SSG available as an AA resource or mentor] – observed in 20 of 67 firms in BSIMM-V and 6 of 130 firms in BSIMM11.
- [CR3.5 Enforce coding standards] – observed in 13 of 51 firms in BSIMM4 and 12 of 130 firms in BSIMM11.

We believe there are two primary reasons why observations for some activities decrease toward zero over time. First, some activities become part of the culture and drive different behavior. For example, the SSG may not need to select satellite members if there is a good stream of qualified volunteers from the engineering groups. Second, some activities don't yet fit tightly with the evolving culture, and doing it currently causes too much friction. For example, continuously going to the SSG for design discussions might unacceptably delay key value streams.

It may also be the case that evolving SSI and DevOps architectures are changing the way some activities are getting done. For example, 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 as a measure of acceptable code.

Just as a point of culture-driven contrast, we see significant increases in observation counts for activities such as [SE2.5 Use application containers], [SE3.5 Use orchestration for containers and virtualized environments], and [SM2.1 Publish data about software security internally] likely for similar reasons that we see lower counts for the activities above. The culture has shifted to be more self-service in engineering and to increased telemetry that produces more data for everyone.

We, of course, keep a close watch on the BSIMM data pool and will make adjustments if and when the time comes, which might include dropping an activity from the model.

Fifty-three of the current participating firms have been through at least two assessments, allowing us to study how their initiatives have changed over time. Twenty-one firms have undertaken three BSIMM assessments, nine have done four, and two have had five assessments.

BSIMM10 was our first study to formally reflect software security changes driven by engineering-led efforts, meaning efforts originating bottom-up in the development and operations teams (often while they evolve into a DevOps group) rather than originating top-down from a centralized SSG. These results showed up here in the form of new activities, in new examples of the way existing activities are conducted, as well as in discussion of the paths organizations might follow to maturation over time. We expanded that analysis for BSIMM11.

BSIMM11 also includes a brief "Trends" section that describes shifts in SSI behavior affecting activity implementation across multiple practices. Larger in scope than an activity, or even a capability that combines activities within a workflow, we believe these trends inform the way that initiatives execute groups of activities within their evolving culture. For example, it's clear that—within engineering-led initiatives particularly—there's a trend toward collecting event-driven security telemetry in addition to or even rather than conducting point-in-time scans that produce reports. This trend fundamentally changes the way organizations interpret and implement multiple activities within the Governance, SSDL Touchpoints, and Deployment domains.

"It's clear that—within engineering-led initiatives particularly—there's a trend toward collecting event-driven security telemetry in addition to or even rather than conducting point-in-time scans that produce reports."

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.

EXECUTIVE LEADERSHIP

Historically, security initiatives that achieve firm-wide impact are sponsored by a senior executive who creates an SSG where software security testing and operations are distinctly separate from software delivery. The BSIMM empowers these individuals to garner resources and provide political support while maturing their groups. Those security initiatives born within engineering and led solely by development management, by comparison, have historically had little lasting impact firm-wide. Likewise, initiatives spearheaded by resources from an existing network security group usually run into serious trouble when it comes time to interface with development groups.

By identifying a senior executive and putting them in charge of software security directly, the organization can address two "Management 101" concerns: accountability and empowerment. It can also create a place where software security can take root and begin to thrive. Whether a firm's current SSI exists primarily in an engineering group or is centrally managed, the BSIMM serves a common purpose by providing leaders insight into the activities firms like their own have adopted and institutionalized. While vendors' marketing and conference submission cycles generate a wealth of new ideas to try, the BSIMM study serves to reduce the guesswork necessary to separate durable activities from clever fads.

SSI Leadership

Individuals in charge of day-to-day efforts in the SSIs we studied have a variety of titles. Examples include:

AppSec Team Lead • Director AppSec • Director AppSec Architecture • Director Cybersecurity
Director Data and Software Security Architecture • Director IT Risk Management & Security
Director IT Shared Services • Director Product Assurance • Director Security Assessment Services
Head AppSec Architecture & Engineering • Head Product Security • Manager AppSec
Manager Vulnerability Management • Product Manager AppSec • Portfolio Manager - Security
Security Program Manager • Senior Director GRC • Senior Director Product Security
Senior Director Shared Services • Senior Manager Operations Security • VP InfoSec
VP Infrastructure and App Risk Management • VP Security Engineering • VP Software

We observe a fairly widespread in exactly where the SSG is situated. In particular:

- 67 of the 130 participating firms have SSGs that are run by a Chief Information Security Officer (CISO) or report to a CISO as their nearest senior executive.
- 21 of the firms report through a Chief Technology Officer (CTO) as their closest senior executive.
- 6 report through a Chief Information Officer (CIO).
- 8 report through a Chief Security Officer (CSO).
- 4 report through a Chief Operations Officer (COO).
- 2 report through a Chief Risk Officer (CRO).
- 1 reports through a Chief Assurance Officer (CAO).
- 1 reports through a Chief Privacy Officer (CPO).
- 20 report to some type of technology or product organization (e.g., Technology, Products & Services, R&D, Engineering, Enterprise Technology).

Of course, not all people with the same title perform, prioritize, enforce, or otherwise provide resources for the same efforts the same way across various organizations.

The significant number of SSGs reporting through a technology executive, in addition to those reporting through the CTO, might reflect the growth of engineering-led initiatives chartered with "building security in" to the software delivery process rather than existing within a compliance-centric mandate.

Organizational alignment for software security is evolving rapidly, and there are constant natural reorganizations over time, so we look forward to next year's data.

In BSIMM-V, we saw CISOs as the nearest executive in 21 of 67 organizations (31.3%), which grew in BSIMM6 to 31 of 78 firms (39.7%), and again for BSIMM7 with 52 of 95 firms (54.7%). Since then, the percentage has remained relatively flat at 54.1% (59 of 109 in BSIMM8), 52.5% (63 of 120 firms in BSIMM9), 58.2% (71 of 122 firms in BSIMM10), and 51.5% (67 of 130 firms in BSIMM11).

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 carried out or facilitated by the software security group (SSG) as part of a practice. Activities are divided into three levels in the BSIMM based on observation rates.*

Champion. *Interested and engaged developers, architects, software managers, testers, and people in similar roles who have a natural affinity for software security and contribute to the security posture of the organization and its software.*

Domain. *One of the four categories our framework is divided into, i.e., Governance, Intelligence, Secure Software Development Lifecycle (SSDL) Touchpoints, and Deployment.*

Practice. *BSIMM activities are organized into 12 categories or practices. Each of the four domains in the software security framework (SSF) has three practices.*

Satellite. *A group of individuals, often called champions, that is organized and leveraged by a software security group (SSG).*

Secure software development lifecycle (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. As software security initiatives (SSIs) evolve, 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. According to our observations, the first step of an SSI is to form an SSG. (Note that, for BSIMM11, we have expanded the definition of the SSG, a fundamental term in the BSIMM world, from implying that the group is always centralized in corporate to specifically acknowledging that the group may be a federated collection of people in corporate, engineering, and perhaps elsewhere. When reading this document and especially when adapting the activities for use in a given organization, use this expanded definition.)*

Software security initiative (SSI). *An organization-wide program to instill, measure, manage, and evolve software security activities in a coordinated fashion. Also known in some literature as an Enterprise Software Security Program.*

SOFTWARE SECURITY GROUP (SSG)

The second most important role in an SSI after the senior executive is the SSG leader and the SSG itself. Each of the 130 initiatives we describe in BSIMM11 has an SSG—a true organizational group dedicated full-time to software security. In fact, without an SSG, successfully carrying out BSIMM activities across a software portfolio is very unlikely, so the creation of an SSG is a crucial first step in working to adopt BSIMM activities. The best SSG members are software security people, but they are often hard to find. If an organization must create a software security team from scratch, it seems best to start with developers and teach them about security. Starting with IT security engineers and attempting to teach them about software, compilers, SDLCs, bug tracking, and everything else in the software universe usually fails to produce the desired results. Unfortunately, no amount of traditional security knowledge can overcome a lack of experience building software.

"SSGs are often asked to mentor, train, and work directly with hundreds of developers, so communication skills, teaching ability, and practical knowledge are must-haves for at least some of the SSG staff."

SSGs come in a variety of shapes and sizes, but SSGs in the more mature SSIs appear to include both people with deep coding experience and people with architectural skill. Code review is an important best practice, but to perform code review, the team must actually understand code (not to mention the huge piles of security bugs therein). That said, the best code reviewers sometimes make poor software architects, and asking them to perform an architecture risk analysis will fail to produce useful findings.

The people on the SSG team

SSGs are often asked to mentor, train, and work directly with hundreds of developers, so communication skills, teaching ability, and practical knowledge are must-haves for at least some of the SSG staff. As the technology landscape changes, leading SSGs make a point of maintaining their depth in evolving disciplines such as digital transformation, cloud infrastructure, CI/CD, DevOps, DevSecOps, privacy, supply chains, and so on. Finally, SSGs are groups of people—whether one person, 10, or 100—who must improve the security posture of the software portfolio, so management skills, risk management perspectives, an ability to contribute to engineering value streams, and an ability to break silos are critical success factors.

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

- Organized to provide software security services.
- Organized around setting policy.
- Designed to mirror business unit organizations.
- Organized with a hybrid policy and services approach.
- Structured around managing a network of others doing software security work.

Some SSGs are highly distributed across a firm whereas others are centralized. Even within the most distributed organizations, we find that product security activities are coordinated by an SSG, even if that SSG is staffed by a single leader with a title such as Security Program Manager or Product Security Manager.

SSG team dynamics and structures

When we look across all the SSGs in our study, we do see several common SSG teams or communities:

- People dedicated to policy, strategy, and metrics.
- Internal services groups that (often separately) cover tools, penetration testing, and middleware development plus shepherding.
- Incident response groups.
- Groups responsible for training development and delivery.
- Externally-facing marketing and communications groups.
- Vendor management groups.
- Groups responsible for security efforts within CI/CD pipelines, clouds, and DevOps teams.

> *"In the spirit of agility, these individuals are often contributing significant amounts of code to delivery, including code that operates continuous build and integration, and code that can go beyond simply adding defect discovery steps to the pipeline."*

As more firms emphasize software delivery speed and agility, whether under the cultural aegis of DevOps or not, we're increasingly seeing SSG structures manifest organically within software teams themselves. Within these teams, the individuals focused on security conduct activities along the critical path to delivering value to customers. Whether staff are borrowed from corporate security or employed by the engineering team directly, we see individuals taking on roles such as product security engineer or security architect, or possessing functional titles such as Site Reliability Engineer, DevOps Engineer, or similar.

SSG team responsibilities

Within these roles, individuals bear responsibilities that are often much greater than that of the more commonplace security champion, including comparison and selection of security tools, definition of secure design guidelines and acceptable remediation actions, and so on. In the spirit of agility, these individuals are often contributing significant amounts of code to delivery, including code that operates continuous build and integration, and code that can go beyond simply adding defect discovery steps to the pipeline. They could even implement security features or broader security architecture as part of the delivery team, as well as author orchestration or other infrastructure-as-code for secure packaging, delivery, and operations. These engineers also contribute monitoring and logging code that aids operations security and incident response. Whereas traditional security champions typically act tactically to execute activities defined by a central SSG, these embedded product security engineers (regardless of moniker) tend to establish and implement, then align other development teams to their improvements, acting in a sense as a distributed SSG.

Here are some SSG-related statistics across the 130 BSIMM11 firms, but note that a handful of large outliers affect the numbers this year:

- We noted an average ratio of full-time SSG members to developers of 2.01%, meaning we found 1 SSG member for every 50 developers when we averaged the ratios of each participating firm. However, the median ratio of full-time SSG members to developers is 0.63% (1 SSG person for each 159 developers), which is probably a more useful number for comparison.
- For organizations with 500 developers or fewer, the largest ratio observed was 51.4% and the smallest was 0.4%, with a median of 1.5%.
- For organizations with more than 500 developers, the largest ratio observed was 3.0% and the smallest was 0.1%, with a median of 0.45%.
- Average SSG size among the 130 firms is 13.9 people (smallest 1, largest 160, median 7). It seems that while SSGs can scale with development size in smaller organizations, the ability to scale quickly drops off in larger organizations (see Table 2).

THE SOFTWARE SECURITY GROUP	
SSG SIZE FOR 130 BSIMM11 FIRMS	Average is 13.9, largest is 160, smallest is 1, median is 7
SSG MEMBER-TO-DEVELOPER RATIO FOR 130 BSIMM11 FIRMS	Average of 2.01%, median of 0.63%
SSG MEMBER-TO-DEVELOPER RATIO FOR BSIMM11 FIRMS WITH FEWER THAN 500 DEVELOPERS	Largest is 51.4%, smallest is 0.4%, median is 1.5%
SSG MEMBER-TO-DEVELOPER RATIO FOR BSIMM11 FIRMS WITH MORE THAN 500 DEVELOPERS	Largest is 3.0%, smallest is 0.1%, median is 0.45%

Table 2. The Software Security Group. Statistical data on SSG size helps everyone put their efforts in perspective.

SATELLITE

In addition to the SSG, many SSIs have identified a number of individuals (often developers, testers, and architects) who share an interest in improving software security but are not directly employed in the SSG. When these individuals carry out software security activities, we collectively refer to them as the satellite. Many organizations refer to this group as their software security champions.

Satellite members are sometimes chosen for software portfolio coverage, with one or two members in each product group, but are sometimes chosen for other reasons such as technology stack coverage or geographical reach. Sometimes, they're more focused on specific issues such as cloud migration and IoT architecture. We are also beginning to see some organizations use satellite members to bootstrap the "Sec" functions they require for transforming a given product team from DevOps to DevSecOps.

Satellite evolution appears to be moving along similar lines with SSI evolution. In many organizations, satellite members are likely to self-select into the group to bring their particular expertise to a broader audience. These individuals often have (usually informal) titles such as CloudSec, OpsSec, ContainerSec, and so on, with a role that actively contributes security solutions into engineering processes. These solutions are often infrastructure-as-code and governance-as-code in the form of scripts, sensors, telemetry, and other friction-reducing efforts.

In any case, the more successful satellite groups get together regularly to compare notes, learn new technologies, and expand stakeholder understanding of the organizations' software security state. Similar to a satellite, and mirroring the community and culture of open source software, we are seeing an increase in motivated individuals in engineering-led organizations sharing digital work products such as sensors, code, scripts, tools, and security features, rather than, for example, getting together to discuss enacting a new policy. Specifically, these engineers are working bottom-up and delivering software security features and awareness through implementation regardless of whether guidance is coming top-down from a traditional SSG.

To achieve scale and coverage, identifying and fostering a strong satellite is important to the success of many SSIs (but not all of them). Some BSIMM activities target the satellite explicitly. Of particular interest:

"The more successful satellite groups get together regularly to compare notes, learn new technologies, and expand stakeholder understanding of the organizations' software security state."

- 13 of the 15 firms with the highest BSIMM scores have a satellite, with an average satellite size of 269 people for those 15 firms.
- Outside the top 15 firms, 30 of the next 50 firms in rank order have a satellite, with an average satellite size of 42.5 for those 30 firms.
- Of the 65 firms with the lowest BSIMM scores, only 19 have a satellite, with the bottom 13 having no satellite at all.

Sixty-four percent of firms that have been assessed more than once have a satellite, while 65% 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. These data suggest that as an SSI matures, its activities become distributed and institutionalized into the organizational structure, and perhaps into engineering automation as well, requiring an expanded satellite to provide expertise and be the local voice of the SSG. Among our population of 130 firms, initiatives tend to evolve from centralized and specialized in the beginning to decentralized and distributed (with an SSG at the core orchestrating things).

EVERYBODY ELSE

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

- Builders, including developers, architects, and their managers, must practice security engineering, taking at least some responsibility for both the definition of "secure enough" as well as ensuring what's delivered achieves that desired posture. Increasingly, an SSI reflects collaboration between the SSG and these engineering-led teams, coordinating to carry out the activities described in the BSIMM.
- Testers typically conduct functional and feature testing, but some progress to include a bit of security testing in their test plans. More recently, some testers are beginning to anticipate how software architectures and infrastructures can be attacked and are working to find an appropriate balance between implementing automation and manual testing to ensure adequate security testing coverage.
- Operations teams must continue to design, defend, and maintain resilient environments. As you will see in the Deployment domain of the SSF, 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 in the operational environment. This means an increasing amount of security effort is becoming software-defined and happening in operations.
- Administrators must understand the distributed nature of modern systems, create and maintain secure builds, and begin to practice the principle of least privilege, especially when it comes to the applications they host or attach to as services in the cloud.

- Executives and middle management, including business owners and product managers, must understand how early investment in security design and security analysis affects the degree to which users will trust their products. Business requirements should explicitly address security needs, including security-related compliance needs. 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 digital transformation efforts that directly improve software security and must actively support digital transformation efforts related to infrastructure and governance-as-code. For a study of CISOs and their organizations, see https://www.synopsys.com/software-integrity/resources/analyst-reports/ciso.html.
- Vendors, including those who supply on-premise products, custom software, and software-as-a-service, are increasingly subjected to service-level agreements (SLAs) and reviews (such as the upcoming Payment Card Industry [PCI] Secure Software Lifecycle Standard and the BSIMMsc) that help ensure products are the result of an SSDL. Of course, vendor or not, the open source management process also requires close attention.

More on Builders and Testers

Builders

Traditionally, as an organization matures, the SSG works to empower builders so they can carry out most BSIMM activities themselves, with the SSG helping in special cases and providing oversight, such as with integrating various defect discovery methods into CI/CD toolchains. In some percentage of engineering-led efforts, the SSG role has changed dramatically, and it serves to publish and spread the activity that engineering-led stakeholders plan, implement, and improve of their own accord.

Today, especially in engineering-led organizations, builders often invite SSG members to participate directly in engineering processes to deliver software and other empowering value rather than written policy and rules. In some cases, the builders are creating their own security microcosms and pressing forward when SSG involvement might be too cumbersome. We often don't explicitly point out whether a given activity is to be carried out by the SSG, developers, or testers. Each organization should come up with an approach that makes sense and accounts for its own workload and software lifecycles.

Testers

Be it test cases driven by a framework such as Cucumber or Selenium to simulate denial of service, by malformed and malicious input, or by more strategic testing of failure through frameworks like ChaosMonkey, some testing regimes are beginning to incorporate nontrivial security test cases. Facilities provided by cloud service providers actively encourage consideration of failure and abuse across the full stack of deployment, such as a microservice component, a data center, or even an entire region going dark.

Similarly, quality assurance practices will have to consider how systems are configured and deployed, for example, by testing configurations for virtualization and cloud components. In many organizations today, software is built in anticipation of failure, and the associated test cases go directly into regression suites run by QA groups or run directly through automation. Increasingly, testers participate in feedback loops with operations teams to understand runtime failures and continuously improve test coverage.

MEASURING YOUR FIRM WITH THE BSIMM

The most important use of the BSIMM is as a measuring stick to determine where your approach currently stands relative to other firms. A direct comparison of all 121 activities is perhaps the most obvious use of the BSIMM. You can simply note which activities you already have in place, find them in the SSF, and then build your scorecard and compare it to the BSIMM11 Scorecard.

As a summary, the BSIMM SSF comprises four domains—Governance, Intelligence, SSDL Touchpoints, and Deployment. In turn, those four domains include 12 practices, which contain the 121 BSIMM11 activities. A BSIMM scorecard indicates which activities were observed in an organization.

On page 22, Table 3 depicts an example firm that performs 40 BSIMM activities (noted as 1s in its scorecard columns), including 10 activities that are the most common in their respective practices (gray boxes). Note the firm does not perform the most commonly observed activities in the other two practices (teal boxes) and should take some time to determine whether these are necessary or useful to its overall SSI. The BSIMM11 AllFirms columns show the number of observations (currently out of 130) for each activity, allowing the firm to understand the general popularity of an activity among the 130 BSIMM11 firms.

Once you have determined where you stand with activities, you can devise a plan to enhance practices with other activities, or perhaps scale current activities across more of the software portfolio. By providing actual measurement data from the field, the BSIMM makes it possible to build a long-term plan for an SSI and track progress against that plan. Note that there's no inherent reason to adopt all activities in every level for each practice. Adopt the ones that make sense for your organization and ignore those that don't but revisit those choices periodically. Once they've adopted an activity set, most organizations then begin to work on the depth, breadth, and cost-effectiveness (e.g., via automation) of each activity in accordance with their view of the associated risk.

"Once you have determined where you stand with activities, you can devise a plan to enhance practices with other activities, or perhaps scale current activities across more of the software portfolio."

In our work using the BSIMM to assess initiatives, we found that creating a spider chart yielding a high-water mark (based on the three levels per practice) is sufficient to obtain a low-resolution feel for maturity, especially when working with data from a particular vertical. We assign the high-water mark with a simple algorithm: if we observed a level 3 activity in a given practice, we assign it a "3" without regard for whether any level 2 or 1 activities were also observed. We assign a high-water mark of 2, 1, or 0 similarly.

One meaningful use of a spider chart comparison is to chart your own high-water mark against the graphs we've published to see how your initiative stacks up. In Figure 1, we have plotted data from the example firm against the BSIMM AllFirms data.

Recall that the breakdown of activities into levels for each practice reflects only the observation frequency for the activities. As such, the levels might illustrate a natural progression through the activities associated with each practice, but it isn't necessary to carry out all activities in a given level before moving on to activities at a different level (activities that are less commonly observed) in the same practice. That said, the levels we use hold water under statistical scrutiny. Level 1 activities (often straightforward and universally applicable) are those that are most commonly observed, level 2 (often more difficult to implement and requiring more coordination) are slightly less frequently observed, and level 3 activities (usually more difficult to implement and not always applicable) are more rarely observed. For many organizations, maturity improves directly as a result of scaling activity efforts across more of the software portfolio and the stakeholders as opposed to aiming specifically at implementing level 3 activities just because they're level 3.

GOVERNANCE			INTELLIGENCE			SSDL TOUCHPOINTS			DEPLOYMENT		
ACTIVITY	BSIMM11 FIRMS (out of 130)	EXAMPLE FIRM	ACTIVITY	BSIMM11 FIRMS (out of 130)	EXAMPLE FIRM	ACTIVITY	BSIMM11 FIRMS (out of 130)	EXAMPLE FIRM	ACTIVITY	BSIMM11 FIRMS (out of 130)	EXAMPLE FIRM
STRATEGY & METRICS			ATTACK MODELS			ARCHITECTURE ANALYSIS			PENETRATION TESTING		
[SM1.1]	94	1	[AM1.2]	81		[AA1.1]	114	1	[PT1.1]	114	
[SM1.2]	70		[AM1.3]	38		[AA1.2]	41	1	[PT1.2]	100	1
[SM1.3]	75	1	[AM1.5]	57	1	[AA1.3]	32	1	[PT1.3]	89	1
[SM1.4]	117	1	[AM2.1]	12		[AA1.4]	67		[PT2.2]	30	
[SM2.1]	64		[AM2.2]	10	1	[AA2.1]	23		[PT2.3]	29	
[SM2.2]	61		[AM2.5]	16	1	[AA2.2]	24	1	[PT3.1]	21	1
[SM2.3]	55		[AM2.6]	10		[AA3.1]	11		[PT3.2]	10	
[SM2.6]	65		[AM2.7]	14		[AA3.2]	1				
[SM3.1]	27		[AM3.1]	3		[AA3.3]	6				
[SM3.2]	4		[AM3.2]	4							
[SM3.3]	18		[AM3.3]	4							
[SM3.4]	1										
COMPLIANCE & POLICY			SECURITY FEATURES & DESIGN			CODE REVIEW			SOFTWARE ENVIRONMENT		
[CP1.1]	94	1	[SFD1.1]	102	1	[CR1.2]	79	1	[SE1.1]	73	
[CP1.2]	112	1	[SFD1.2]	76	1	[CR1.4]	100	1	[SE1.2]	121	1
[CP1.3]	86	1	[SFD2.1]	32		[CR1.5]	49		[SE2.2]	39	1
[CP2.1]	55		[SFD2.2]	51		[CR1.6]	41	1	[SE2.4]	33	
[CP2.2]	47		[SFD3.1]	14		[CR1.7]	50		[SE2.5]	31	1
[CP2.3]	61		[SFD3.2]	14		[CR2.6]	23	1	[SE2.6]	36	1
[CP2.4]	48		[SFD3.3]	4		[CR2.7]	24		[SE3.2]	13	
[CP2.5]	70	1				[CR3.2]	7		[SE3.3]	7	
[CP3.1]	26					[CR3.3]	3		[SE3.5]	22	1
[CP3.2]	16					[CR3.4]	2		[SE3.6]	12	
[CP3.3]	8					[CR3.5]	1				
TRAINING			STANDARDS & REQUIREMENTS			SECURITY TESTING			CONFIG. MGMT. & VULN. MGMT.		
[T1.1]	83	1	[SR1.1]	93	1	[ST1.1]	104	1	[CMVM1.1]	108	1
[T1.5]	38		[SR1.2]	90		[ST1.3]	89	1	[CMVM1.2]	102	
[T1.7]	53	1	[SR1.3]	94	1	[ST2.1]	42		[CMVM2.1]	94	1
[T1.8]	41		[SR2.2]	64	1	[ST2.4]	20		[CMVM2.2]	97	
[T2.5]	32		[SR2.4]	60		[ST2.5]	16		[CMVM2.3]	65	
[T2.8]	28	1	[SR2.5]	44	1	[ST2.6]	13		[CMVM3.1]	5	
[T3.1]	4		[SR3.1]	30		[ST3.3]	5		[CMVM3.2]	11	
[T3.2]	22		[SR3.2]	9		[ST3.4]	2		[CMVM3.3]	13	
[T3.3]	16		[SR3.3]	9		[ST3.5]	2		[CMVM3.4]	16	
[T3.4]	19		[SR3.4]	25		[ST3.6]	0		[CMVM3.5]	8	1
[T3.5]	7								[CMVM3.6]	0	
[T3.6]	1										

LEGEND		
	ACTIVITY	121 BSIMM11 activities, shown in 4 domains and 12 practices
	BSIMM11 FIRMS	Count of firms (out of 130) observed performing each activity
	⬜	Most common activity within a practice
	🟦	Most common activity in practice was not observed in this assessment
	🟨 1	Most common activity in practice was observed in this assessment
	⬛	A practice where firm's high-water mark score is below the BSIMM11 average

Table 3. BSIMM11 ExampleFirm Scorecard. A scorecard is helpful for understanding efforts currently underway. It also helps visualize the activities observed by practice and by level to serve as a guide on where to focus next.

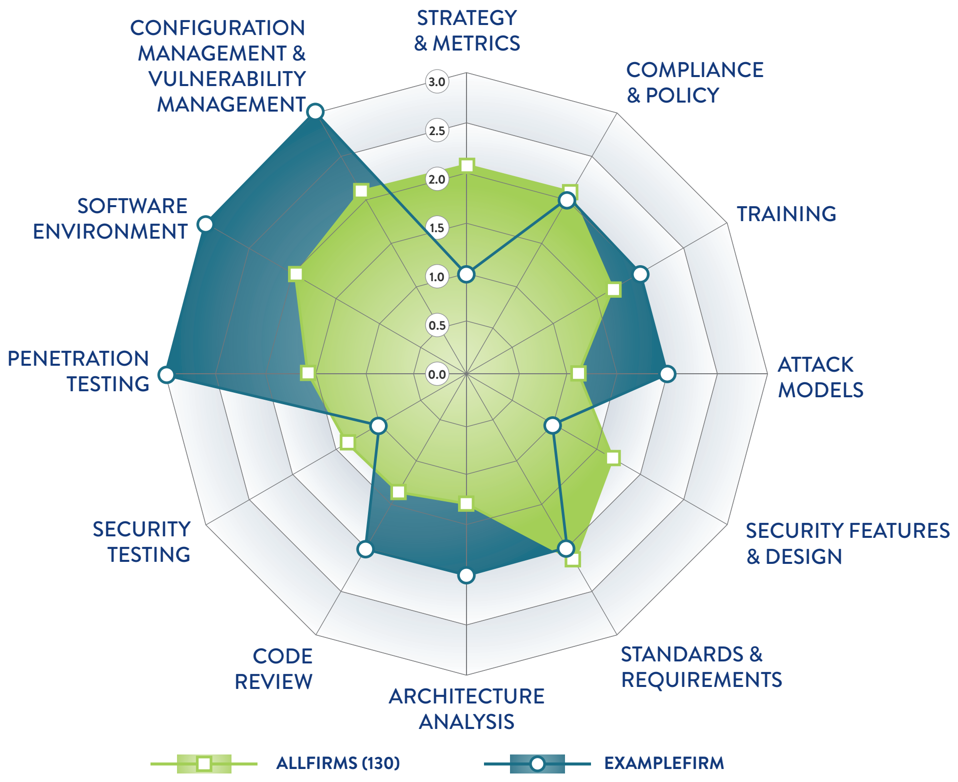


Figure 1. AllFirms vs. ExampleFirm Spider Chart. Charting high-water mark values provides a low-resolution view of maturity that can be useful for comparisons between firms, between business units, and within the same firm over time.

By identifying activities from each practice that are useful for you, and by ensuring proper balance with respect to domains, you can create a strategic plan for your SSI moving forward. Note that most SSIs are multiyear efforts with real budget, mandate, and ownership behind them. In addition, while all initiatives look different and are tailored to fit a particular organization, all initiatives share common core activities (see "Table 5. Most Common Activities Per Practice" in Part Three).

USING THE BSIMM TO START OR IMPROVE AN SSI

The BSIMM is not 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 from the ground up and when evolving their initiatives through various maturity phases over time.

SSI PHASES

No matter an organization's culture, all firms strive to reach similar waypoints on their software security journey. Over time, we find that SSIs typically progress through three states:

- **Emerging**. An organization tasked with booting a new SSI from scratch or formalizing nascent or ad hoc security activities into a holistic strategy. An emerging SSI has defined its initial strategy, implemented foundational activities (e.g., such as those observed most frequently in each practice), acquired some resources, and might have a roadmap for the next 12 to 24 months of its evolution. SSI leaders working on a program's foundations are often resource-constrained on both people and budget, and might create a small SSG that uses compliance requirements or other executive mandates as the initial drivers to continue adding activities.

- **Maturing**. An organization with an existing or emerging software security approach connected to executive expectations for managing software security risk and progressing along a roadmap for scaling security capabilities. A maturing SSI often works to apply existing activities to a greater percentage of the firm's technology stacks, software portfolio, and engineering teams (both in-house and supply chain). SSI leadership trying to mature a program might mean adding fewer activities while working on depth, breadth, and cost-effectiveness of ongoing activities and building bridges with stakeholders in the business and in engineering.
- **Optimizing**. An organization that's fine-tuning and evolving its existing security capabilities (often with a risk-driven approach), having a clear view into operational expectations and associated metrics, adapting to technology change drivers, and demonstrating risk management and business value as differentiators. The SSI leader optimizing their program might also be undergoing personal growth from technology executive to business enabler.

"Experience shows that SSIs can reach a 'maturing' stage by starting with the activities that are right for them without regard for the total activity count."

It's compelling to imagine that organizations could self-assess and determine that by doing X number of activities, they qualify as emerging, maturing, or otherwise. However, experience shows that SSIs can reach a "maturing" stage by starting with the activities that are right for them (e.g., to meet external contractual or compliance requirements) without regard for the total activity count. This is especially true when considering software portfolio size and the relative complexity of maturing or optimizing some activities across 1, 10, 100, and 1,000 applications.

In addition, organizations don't always progress from emerging to optimizing in one direction or in a straight path. We have seen SSIs form, break up, and re-form over time, so one SSI might go through the emerging cycle a few times over the years. Moreover, an SSI's capabilities might not all progress through the same states at the same time. 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 optimizing. There is also constant change in tools, skill levels, external expectations, attackers and attacks, resources, culture, and everything else. Pay attention to the relative frequency with which the BSIMM activities are observed across all the participants but use your own metrics to determine if you're making the progress that's right for you.

TRADITIONAL SSI APPROACHES

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

- Organizations where the SSI was purposefully started in a central corporate group (e.g., under a CISO) and focused on compliance, testing, and risk management. This path is seen most often in regulated industries such as banking, insurance, and healthcare but is also seen in some ISV and technology firms.
- Organizations where the SSI was started by engineering leadership (e.g., senior application architects) who then created a centralized group (e.g., under a CTO) to set some development process, create and manage security standards, and ensure the silos of engineering, testing, and operations are aware of and adhere to security expectations. This path is seen most often in technology firms and ISVs but is also seen in other verticals.

Regardless of origin and whether the SSG currently lives in a corporate group or in engineering, both cultures historically ended up with an SSI that is driven by a centralized, dedicated SSG whose function is to ensure appropriate software security activities are happening across the portfolio. That is, nearly all SSIs today are governance-led, regardless of whether the SSI genesis was the executive team or the architecture team. They practice proactive risk management through assurance-based activities, resulting in the creation of "rules that people must follow" and "expectations software must meet" (e.g., policy, standards, checkpoints, sensors). This has almost always resulted in a defined process that includes prescriptive testing at various times in the lifecycle and checkpoints where engineering processes might be derailed.

When software security activities are developed within the context of an engineering team, they are often well-tuned to that team's risk tolerance, which might generate friction with groups like audit, compliance, or even other engineering teams. Where central groups dictate policy and activities that make a lot of risk management sense to the SSG, those requirements often just appear as friction to the development, testing, and operations groups. We perceive neither culture as measurably better but point out that in either case, the SSG must quickly and proactively reach out to other departments, normalize activities across disparate risk and usability tolerances, and socialize the resulting culture organization-wide.

"Regardless of origin and whether the SSG currently lives in a corporate group or in engineering, both cultures historically ended up with an SSI that is driven by a centralized, dedicated SSG whose function is to ensure appropriate software security activities are happening across the portfolio."

THE NEW WAVE IN ENGINEERING CULTURE

We're once again seeing a wave of software security efforts emerging from engineering teams. These teams are usually responsible for either delivering a product or value stream—such as is common within ISVs—or for maintaining a technology domain—such as the "cloud security group" or a part of some digital transformation group. At least two large factors are driving these engineering-led efforts, which have been rapidly expanding over the last few years:

- The confluence of process friction, unpredictable impacts on delivery schedules, adversarial relationships, and a growing number of human-intensive processes from existing SSIs.
- 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.

One imperative common to both cultural and technology shifts is engineer self-service: self-service IT (cloud), self-service configuration and deployment (DevOps), and self-service development (open source use and continuous integration). A natural result of this self-service culture is engineering groups placing more emphasis on automation as opposed to human-driven tasks. Automation in the form of software-defined sensors and checkpoints is removing unexpected variability in human processes but is often also replacing human discretion, conversations, and risk decisions that should involve multiple stakeholders. Many application lifecycle processes are moving faster whether they're ready to or not. And, perhaps most importantly, all this software security effort is frequently happening independently from the experience and lessons learned that a centralized SSG might provide.

The governance-driven approach we've seen for years along with the emerging engineering-driven efforts are increasingly coexisting within the same organization and often have competing objectives, even while pulling traditional governance-driven programs into modern and evolving hybrids. Figure 2 shows this ongoing SSG evolution.

"Automation in the form of software-defined sensors and checkpoints is removing unexpected variability in human processes but is often also replacing human discretion, conversations, and risk decisions that should involve multiple stakeholders."

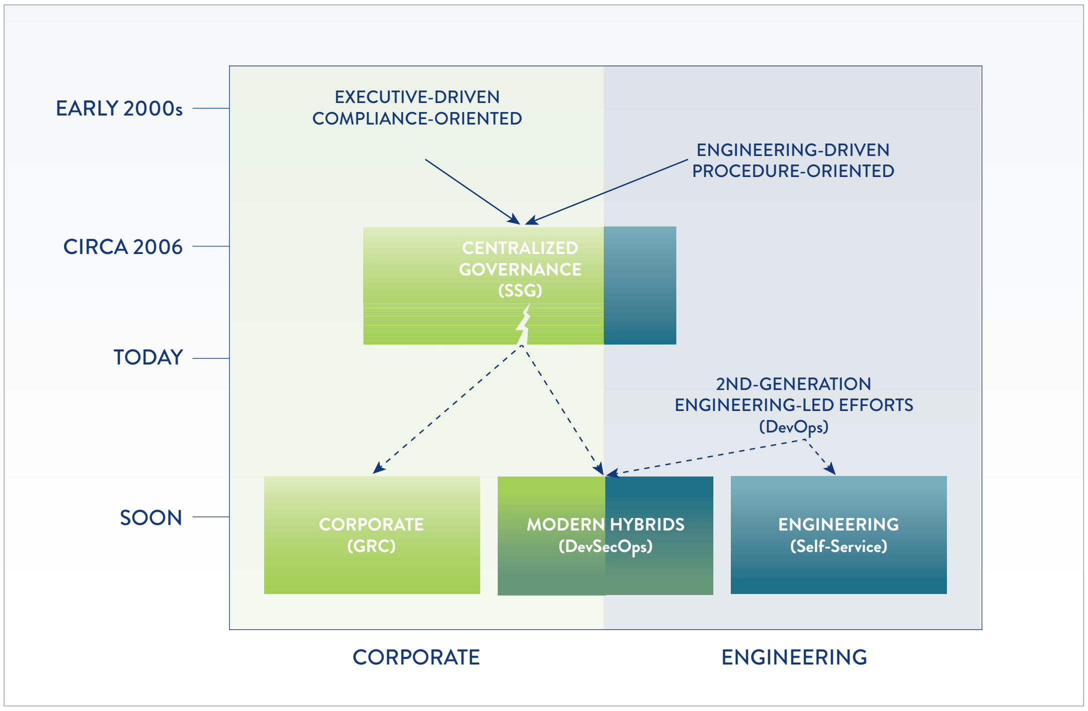


Figure 2. SSG Evolution. *These groups might have started in corporate or in engineering but, in general, settled on enforcing compliance with tools. The new wave of engineering-led efforts is shifting where SSGs live, what they focus on, and how they operate.*

The DevOps movement has put these tensions center stage for SSG leaders to address. Given different objectives, we find that the outcomes desired by these two approaches are usually very different. Rather than the top-down, proactive risk management and "rules that people must follow" 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 (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-led efforts are independently evolving both how they apply software security activities and, in some cases, what activities they apply. The changes engineering-led teams are making include downloading and integrating their own security tools, spinning up cloud infrastructure and virtual assets as they need them, following policy on use of open source software in applications but routinely downloading dozens or hundreds of other open source packages to build and manage software and processes, and so on. Engineering-led 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.

In addition to centralized SSI efforts and engineering-led efforts, cloud service providers, software pipeline and orchestration platforms, and even quality assurance tools have all begun adding their view of security as a first-class citizen of their feature sets, user guides, and knowledge bases. For example, organizations are seeing platforms like GitHub and GitLab beginning to compete vigorously using security as a differentiator, leading both providers to create publicly available security documentation with a 12- to 36-month vision. Evolving vendor-provided features might be signaling to the marketplace and adopting organizations that vendors believe security must be included in developer tools, and that engineering-led security initiatives should feel comfortable relying on these platforms as the basis of their security telemetry and even their governance workflows.

"The most forward-looking engineering-led efforts are making technology-specific changes to their security activities-as-code at the same speed that their development and cloud technology changes."

Given the frequent focus by centralized governance groups on policy adherence and test results as measures of success, such groups have historically not recognized the specific progress made by engineering-led teams. For their part, engineering teams do not frequently broadcast improvements to vulnerability discovery or security engineering until those changes are shareable as reusable code. We observe that the evolution of pipeline platform tools and their security features, including governance workflows and reporting dashboards, is beginning to repair the disconnect.

The most forward-looking engineering-led efforts are making technology-specific changes to their security activities-as-code at the same speed that their development and cloud technology changes. Similarly, centralized SSI owners are making changes to their policies, standards, and processes at the same speed that executives can build consensus around them. Security executives are increasingly acknowledging this mismatch in agility and explicitly identifying the need for all stakeholders to come up to speed on modern developer tooling and culture. Combining these two approaches and cadences, while still maintaining a single coherent SSI direction, will require a concerted effort by all the stakeholders.

CONVERGENCE AS A GOAL

We frequently observe governance-driven SSIs planning centrally, seeking to proactively define an ideal risk posture during their emerging phase. After that, the initial uptake of 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. These firms often struggle during the maturation phase where growth will incur significant expense and effort as the SSG scales the controls and their benefits enterprise-wide.

We also observe that emerging engineering-driven efforts prototype controls incrementally, building on the 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 during their maturation phase, 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.

Scaling an SSI across a software portfolio is hard for everyone. Today's evolving cultural and technological environments seem to require a concerted effort at converging centralized and engineering efforts to create a cohesive SSI that ensures the software portfolio is appropriately protected.

Emerging engineering-driven 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 then 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-driven groups. SSGs, and likely champions as well, that begin to support this approach will likely speed up the convergence of various group efforts and the alignment with corporate risk management goals.

"Governance-led groups often focus on rules, gates, and compliance, while emerging engineering-led efforts usually focus on feature velocity, error-avoidance through automation, and software resilience. Success doesn't require identical viewpoints, but collectively they need to converge in order to keep the firm safe."

Importantly, the delivery pipeline platforms upon which many engineering teams rely have begun to support a broader set of security activities as on-by-default security features. As examples, OpenShift, GitHub, and GitLab are processing external security research, responsibly disclosing vulnerabilities and notifying users, and providing some on-by-default defect discovery, vulnerability management, and remediation management workflows. This allows engineering-driven firms to share success between teams that use these platforms simply by sharing configuration information, thus propagating their security policies quickly and consistently. It also allows SSGs and champions to more easily tie in at key points in the SSDL to perform governance activities with minimal impact on software pipelines.

Though the BSIMM data and our analyses don't dictate specific paths to SSI maturity, we have observed patterns in the ways firms use the activities to improve their capabilities. Governance-led and emerging engineering-led approaches to software security improvement embody different perspectives on risk management that might not correlate. Governance-led groups often focus on rules, gates, and compliance, while emerging engineering-led efforts usually focus on feature velocity, error-avoidance through automation, and software resilience. Success doesn't require identical viewpoints, but collectively they need to converge in order to keep the firm safe. That means the groups must collaborate on risk management concerns to build on their strengths and minimize their weaknesses.

Given these cultural differences, how does one go about building and maturing a firm-wide SSI that protects the entire software portfolio? Aligning governance views and engineering views is a requirement for moving forward effectively. This is an important transition that is still in progress in nearly all organizations.

Every organization is on its own unique software security journey. Evolving business and technology drivers, executive expectations, security goals, and operational aspirations, as well as current organizational strengths and weaknesses, will motivate different paths from your current state to your journey's next waypoint.

A DESCRIPTION OF TWO JOURNEYS

We see these paths as patterns in how SSIs are implemented and trends in how they evolve. Many of these patterns are common to programs, whether they're working on foundations, scale, or efficacy. They are observed across cultural boundaries, even if the security culture is one of central and proactive governance or is engineering-driven, or if both are learning to coexist.

In the two journeys below, we include specific references to BSIMM activities. These references are meant to help you understand associations between the topic being discussed and one or more BSIMM activities. The references don't mean that the topic being discussed is fully equivalent to the activity. For example, when we say, "Inventory software [SM3.1]," we don't mean that having an inventory encompasses the totality of [SM3.1], just that having an inventory will likely be something you'll do on your way to implementing [SM3.1]. To continue using [SM3.1] 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 inventory, implement a process to keep the inventory up to date, find a way to track results from testing efforts, do some repeatable analysis, and decide how to create a risk posture view that's meaningful for them. Every activity has its own nuances and components, and every organizational journey will be unique.

"Evolving business and technology drivers, executive expectations, security goals,and operational aspirations, as well as current organizational strengths and weaknesses, will motivate different paths from your current state to your journey's next waypoint."

Although they might not use the same vocabulary or originate in the same organizational structure, nearly all SSIs build a foundation that includes the following:

- **Structure.** Name an owner [SM1.1], generate awareness [SM1.2], and identify engineering participation [SFD1.2, CMVM1.1].
- **Prioritization.** Define a list of Top 10 bugs [CR2.7] to prevent or attacks [AM2.5] to mitigate, prioritize portfolio scope [AA1.4], select initial checkpoints [SM1.4], define compliance needs [CP1.2], and ensure the basics [SE1.2].
- **Visibility.** Inventory assets [SM3.1, CP2.1, CMVM2.3], then conduct defect discovery [AA1.1, CR1.4, ST2.1, PT1.1, SR2.4] to determine which issues are already in production.

Note that an SSI leader with a young initiative (e.g., less than one year) working on the foundations should not expect or set out to quickly implement a large number of BSIMM activities. Firms can absorb only a limited amount of cultural and process change at any given time. The BSIMM11 data show that SSIs having an age of one year or less at the time of assessment have an average score of 24.7 (24 of 130 firms).

THE GOVERNANCE-LED JOURNEY

Until recently, the history of software security advancement has been largely driven by three groups. One is highly-regulated industries—mostly financial services—striving to both educate regulators and stay ahead of their demands. Another is forward-looking ISV and high-technology firms that worked diligently to strike a balance between governance and engineering agility. Of course, software security experts have also spent a couple of decades helping mature the broad discipline.

GOVERNANCE-LED CULTURE

Leadership

Governance-driven SSIs almost always begin their journey by appointing an SSI owner tasked with shepherding the organization through understanding scope, approach, and priorities. Once an SSI owner is in place, their first order of business is likely to establish a centralized structure. 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 assurance objectives that are further defined and institutionalized in policy [CP1.3], standards [SR1.1], and processes [SM1.1].

Inventory Software

We observe governance-led SSIs seeking an enterprise-wide perspective when building an initial view into their software portfolio. Engaging directly with application business owners, these cultures prefer to cast a wide net using questionnaire-style data gathering to build their initial application inventory [CMVM2.3]. These SSIs tend to focus on applications (with owners who are responsible for risk management) as the unit of measure in their inventory rather than software, which might also include many vital components that aren't applications. In addition to understanding application profile characteristics (e.g., programming language, architecture type such as web or mobile, revenue generated) as a view into risk, these cultures tend to focus on understanding where sensitive data resides and flows (e.g., PII inventory) [CP2.1] along with the status of active development projects.

The nature of inventorying software is evolving dynamically. Simple spreadsheet lists of application names gave way to the need for open source enumeration, a problem still being addressed by many firms. Even with these efforts, many firms are still only inventorying their source code. Expanding inventories to include all binaries, dependencies, and calls to services is more complicated, and accounting for all software running in production is aspirational for most firms.

Select In-Scope Software

With an application inventory in hand, governance-led SSIs impose security requirements top-down using formalized risk-based approaches to blanket as much of their software portfolio as possible. Using simple criteria (e.g., application size, regulatory constraints, internal vs. external facing, data classification), these cultures assign a risk classification (e.g., high, medium, low) to each application in their inventory [AA1.4]. SSI leaders then 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 environment with a native architecture) when selecting foundational SSI efforts. We also observe that firms find it beneficial to include in the selection process some engineering teams that are already doing some security activity organically.

GOVERNANCE-LED CHECKLIST FOR GETTING STARTED

1. **Leadership**. *Put someone in charge of software security and provide the resources they will need to succeed.*
2. **Inventory software.** *Know what you have, where it is, and when it changes.*
3. **Select in-scope software.** *Decide what you're going to focus on first and contribute to its value streams.*
4. **Ensure host and network security basics.** *Don't put good software on bad systems or in poorly constructed networks (cloud or otherwise).*
5. **Do defect discovery.** *Determine the issues in today's production software and plan for tomorrow.*
6. **Engage development.** *Identify those responsible for software delivery pipelines, key design, and code, and involve them in the planning, implementation, and roll-out at scale of security activities.*
7. **Select security controls.** *Start with controls that establish some risk management to prevent recurrence of issues you're seeing today.*
8. **Repeat.** *Expand the team, improve the inventory, automate the basics, do more prevention, and then repeat again.*

Ensure Host and Network Security Basics

One of the most commonly observed activities today regardless of SSG age is [SE1.2 Ensure host and network security basics are in place]. A common strength for governance-minded firms that have tight controls over the infrastructure assets they manage, this is accomplished through a combination of IT provisioning controls, written policy, pre-built and tested golden images, sensors and monitoring capabilities, server hardening and configuration standards, and entire groups dedicated to patching. As firms migrate infrastructure off-premise to cloud environments, governance-led firms remain keen on re-implementing their assurance-based controls to verify adherence to security policy, calling out cloud provider dependencies. They sometimes must deploy custom solutions to overcome limitations in a cloud provider's ability to meet desired policy in an attempt to keep tabs on the growing number of virtual assets created by engineering groups and their automation.

Do Defect Discovery

Initial defect discovery efforts in governance-led cultures tend to be one-off using centralized commercial tools [CR1.2] and tend to target the most critical software with a plan to scale efforts over time. Often, a previous breach or near miss focuses everyone's attention on one particular type of technology stack or security defect. While not always automated or repeatable, conducting some vulnerability discovery in order to get a feel for the current risk posture allows firms to prioritize remediation and motivate the necessary conversations with stakeholders to gain buy-in for an SSG. The type of vulnerability discovery doesn't necessarily matter at this stage and might be selected because it applies to the current phase of the software lifecycle that the intended target is naturally progressing through (e.g., do threat modeling at design time and penetration testing on deployed software for a critical application).

"Enrolling development begins by creating mutual awareness of how the SSI and development teams see the next steps in maturing security efforts."

Engage Development

Engineering teams are likely already thinking about various aspects of security related to software, configuration, infrastructure, and related topics. Enrolling development begins by creating mutual awareness of how the SSI and development teams see the next steps in maturing security efforts. Successfully engaging development early on relies on bridge-building and credentialing the SSG as competent in development culture, toolchains, and technologies, building awareness around what security practices constitute an SSDL, and roughly determining how those practices 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.

Select Security Controls

Based on the kinds of software in inventory, the reasons for selecting certain applications to be within your program's scope, and the issues uncovered in initial defect discovery efforts, SSI leaders select those BSIMM activities directly applicable to incrementally improving the security of their application portfolio (e.g., policy [CP1.3], testing [AA1.2, CR1.4, ST2.1, PT1.3, SR2.4], training [T1.5]) and implement them in a quick-win approach. Governance-minded cultures tend to prefer showing adherence to well-known guidance and often choose initial security controls in response to general industry guidance (e.g., regulators, OWASP, CWE, NIST, analysts) that applies to as much of their software as possible. Initial selection is likely focused on detective controls (e.g., testing) to maximize visibility into the organization's risk posture.

MATURING GOVERNANCE-LED SSIs

With foundations for centralized governance established, SSI 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. Driven centrally and communicated top-down, these cultures prescribe when security activities must occur at each phase of the software lifecycle [SM1.4] and begin onboarding application teams into the SSDL to improve overall risk posture.

Document and Socialize the SSDL

Because requirements come from the top, these firms typically prefer to create process, policy, and security standards in document and presentation form. Specifically, these firms document a process (e.g., prototype SSDL) to generalize the SSI efforts above and communicate it to everyone for the organization's use [SM1.1]. Creating even a single-page view of the defect discovery activities to be conducted allows the organization to institutionalize its initial awareness-building efforts as the first revision of its governance and testing regimen.

We observe governance-minded firms publishing security policies and standards through already established governance, risk, and compliance (GRC) channels, complementing existing IT security standards. The SSI leader can also create a security portal (e.g., website, wiki) that houses SSDL information in a central place [SR1.2]. Similar to the approach for prioritizing defect discovery efforts, we observe these firms driving initial standards creation from industry top N risks, leveraging sources such as MITRE, ISO, and NIST to form baseline requirements [AM2.5, CR2.7].

Finally, in governance-led SSIs, getting the word out about the organization's top N risks and what can be done about them becomes a key part of the SSI leader'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 [SM1.2].

Balance Detective and Preventive Controls

As evidence is gathered through initial defect discovery efforts that highlight the organization's serious security defects in its most important software assets, SSI leaders in governance-led firms seek to balance detective controls with preventive controls for avoiding security issues and changing developer behavior.

"SSI leaders in governance-led firms seek to balance detective controls with preventive controls for avoiding security issues and changing developer behavior."

Because initial defect discovery often targets deployed software (over to the right in the software lifecycle), SSI leaders begin educating the organization on the need to shift left through adoption of tools that can be integrated into developer workflows. These firms typically rely on tool vendor rulesets and vulnerability coverage to expand their generic top N-focused defect discovery, often starting with static [CR1.4] and dynamic analysis [ST2.1, ST2.6] to complement existing penetration testing [PT1.1, PT1.3] efforts. To get started with tool adoption, we observe SSI leaders dedicating some portion of their staff to serve as tool mentors and coaches to help development teams not only integrate the tools but also triage and interpret results for the first time [CR1.7]. Seeking portfolio coverage when evaluating and selecting tools, governance-led SSIs often consider language support, environment interoperability, ease of deployment, and results accuracy as key success criteria. Note that this effort to shift left almost never extends outside the SDLC to requirements and design.

To scale the security mindset as tools are adopted, we observe governance-led firms focusing on establishing security champions within application teams [SM2.3]. Although the primary objective is to embed security leadership inside development, these individuals also serve as key points of contacts and interface points for the SSG to interact with application teams and monitor progress. Because they are local to teams, champions also facilitate defect management goals, such as tracking recurring issues to drive remediation [PT1.2]. To the extent that organizations are beginning to rely more heavily on open source defect discovery tools, champions are serving as a means of implementing these tools for development teams, as well as being the in-team tool mentor.

Similar to policy, the need for tool adoption and building a satellite of champions is often communicated top-down by the SSI leader. Starting at the executive level, SSI leaders often arm the CISO, CTO, or similar executive with data from initial defect discovery efforts to tell stories about the consequences of having insecure software and how the SSDL will help [SM2.1]. At the developer level, SSI leaders begin rolling out foundational software security training material tailored to the most common security defects identified through defect discovery efforts, often cataloged by technology stack [T1.7].

Support Incident Response and Feedback to Development

In some governance-led cultures, establishing a link between the SSG and those doing the monitoring for security incidents often happens naturally when CISOs or similar executives also own security operations. We observe SSI leaders participating in software-related security incidents in a trusted advisor role to provide guidance to operations teams on applying a temporary compensating control (e.g., a short-term web application firewall [WAF] rule) and support to development teams on fixing a root cause (e.g., refactor code, upgrade a library) [CMVM1.1]. For firms that have an emerging satellite, those champions are often pulled into the conversation to address the issue at its source, creating a new interdepartmental bridge between the SSG, security operations, and development teams.

OPTIMIZING GOVERNANCE-LED SSIs

Achieving software security scale—of expertise, portfolio coverage, tool integration, vulnerability discovery accuracy, process consistency, and so on—remains a top priority. However, firms often scale one or two capabilities (e.g., defect discovery, training) but fail to scale other capabilities (e.g., architecture analysis, vendor management). Once scaled, there's a treasure trove of data to be harvested and included in KPI and KRI reporting dashboards. Then, executives start asking 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].

“Organizations don't always progress from emerging to optimizing in one direction or on a straight path, and some SSI capabilities might be optimizing while others are still emerging.”

Progress Isn't a Straight Line

As mentioned earlier, organizations don't always progress from emerging to optimizing in one direction or on a straight path, and some SSI capabilities might be optimizing while others are still emerging. Based on our experience, firms with some portion of their SSI operating in an optimized state have likely been in existence for longer than three years. Although we don't have enough data to generalize this class of initiative, we do see common themes for those who strive to reach to 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 [SR3.1], vendor code [SR3.2], toolchains, or any associated environments and processes [SE1.2, SE2.6, SE3.3]. These top N risks are most useful when specific to the organization, evaluated at least annually, and tied to metrics as a way to prioritize SSI efforts to improve risk posture.
- **Tool customization.** SSI leaders place a concerted effort into tuning tools (e.g., customization for static analysis, fuzzing, penetration testing) to improve integration, accuracy, consistency, and depth of analysis [CR2.6]. Customization focuses not only on improving results fidelity and applicability across the portfolio but also on pipeline integration and timely execution, improving ease of use for everyone.
- **Feedback loops.** Loops are specifically created between SSDL activities to improve effectiveness, as deliverables from SSI capabilities ebb and flow with each other. As an example, an expert within QA might leverage architecture analysis results when creating security test cases [ST2.4]. 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 seems to be gaining ground in some firms.
- **Data-driven governance.** Leaders instrument everything to collect data that in turn become metrics for measuring SSI efficiency and effectiveness against KRIs and KPIs [SM3.1]. 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 evolve over time.

Push for Agile-Friendly SSIs

In recent years, we've observed governance-led firms—often out of necessity to remain in sync with development changes—evolving to become more agile-friendly:

- Putting "Sec" in DevOps is becoming a mission-critical objective. SSI leaders routinely partner with IT, cloud, development, quality assurance, and operations leadership to ensure the SSG mission aligns with DevOps values and principles.
- SSI 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. We expect this list to grow as other topics become more mainstream, such as serverless computing and single-page application languages.
- To align better with DevOps values (e.g., agility, collaboration, responsiveness), SSI leaders are beginning to replace traditional people-driven activities with people-optional, pipeline-driven automated tasks. Often this 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 in development and deployment streams (e.g., security sensors) to directly support SSI activities.
- SSI 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.

THE ENGINEERING-LED JOURNEY

From an activity perspective, we observe that emerging engineering-led software security efforts build on a foundation very similar to governance-led organizations. How they go about accomplishing these activities differs and usually parallels their software-delivery focus.

EMERGING ENGINEERING-LED CULTURE

Inventory Software

One of the first activities is creating an initial inventory of their local software portfolio [CMVM2.3]. Rather than taking an organizational structure and owner-based view of this problem, we observe emerging engineering-led efforts attempting to understand software inventory by extracting it from the same tools they use to manage their IT assets. By scraping these software and infrastructure configuration management databases (CMDBs), they craft an inventory brick-by-brick rather than top-down. They use the metadata and tagging that these content databases provide to reflect their software's architecture as well as their organization's structure.

ENGINEERING CHECKLIST FOR GETTING STARTED

1. **Inventory software.** *Know what you have, where it is, and when it changes.*
2. **Select in-scope software.** *Decide what you're going to focus on first and contribute to its value streams.*
3. **Ensure host and network security basics.** *Don't put good software on bad systems or in poorly constructed networks (cloud or otherwise).*
4. **Choose application controls.** *Apply controls that deliver the right security features and also help prevent some classes of vulnerabilities.*
5. **Repeat.** *Expand the team, improve the inventory, automate the basics, do more prevention, and then repeat again.*

To this end, engineering-led efforts can combine two or more of the following approaches to 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.

That last bullet is particularly interesting. We've observed organizations learning that this kind of external discovery is essential, despite substantial efforts to develop or purchase means of internal discovery such as described by the first two bullets. In other words, despite increasing efforts, many organizations have substantially more software in production environments today than is captured by their existing inventory processes. As one simple example, does a monolithic web application replaced by a smaller application and 25 microservices become 26 entries in your CMDB? When the answer is no, all organizations struggle to find all their software after the fact.

Select In-Scope Software

We observe security leadership within engineering-led security efforts informally prioritizing in-scope software rather than using a published and socialized risk formula. As software solutions pivot to meet changing customer demand, the list of software that is in scope of security governance is likely more fluid than in governance-driven groups. In engineering-led efforts, informal prioritization that is revisited with much greater frequency helps them better respond and prioritize the appropriate software changes.

For much of an engineering-led effort, an activity is first prototyped by a security engineer participating within a software delivery team. These engineers individually contribute to a team's critical path activities. When they, for example, complete production of test automation [ST2.5], vulnerability discovery tooling, or security features [SFD1.1], one of the security engineers might move onto another delivery team, bringing along their accomplishments, or they might seek to gain leverage from those accomplishments through the organization's knowledge management systems and personally evangelize their use. This level of intimacy between a developer and the security improvements they spread to other projects and teams—often regardless of whether that improvement closely aligns with governance-driven rules—makes scoping and prioritizing stakeholder involvement in the software inventory process vitally important.

"In engineering-led efforts, informal prioritization that is revisited with much greater frequency helps them better respond and prioritize the appropriate software changes."

Prioritizing In-Scope Software

Drivers differ by organization, but engineering-led groups have been observed using the following as input when prioritizing in-scope software:

- **Velocity.** Teams conducting active new development or major refactoring.
- **Regulation.** Those services or data repositories to which specific development or configuration requirements for security or privacy apply [CP1.1, CP1.2].
- **Opportunity.** Those teams solving critical technical challenges or adopting key technologies that potentially serve as proving grounds for emerging security controls.

Beyond immutable constraints like the applicability of regulation, we see evidence that assignment can be rather opportunistic and perhaps driven bottom-up by security engineers and development managers themselves. In these cases, the security initiative's leader often seeks opportunities to cull their efforts and scale key successes rather than direct the use of controls top-down.

Ensure Host and Network Security Basics

Compared to governance-led organizations, [SE1.2 Ensure host and network security basics are in place] is observed no less frequently for engineering-led groups. Security engineers might begin by conducting this work manually, then baking these settings and changes into their software-defined infrastructure scripts to ensure both consistent application within a development team and scalable sharing across the organization.

Forward-looking organizations that have adopted software and network orchestration technologies (e.g., Kubernetes, Envoy, Istio) get maximum impact from this activity with the efforts of even an individual contributor, such as a security-minded DevOps engineer. While organizations often have hardened container or host images on which software deployments are based, software-defined networks and features from cloud service providers allow additional control at the scale of infrastructure.

Though many of the technologies in which security engineers specify hardening and security settings are human-readable, engineering-led groups don't typically take the time to extract and distill a document-based security policy from these codebases. Without such policy that's easy to consume by nontechnical humans, it can be hard for centralized elements of growing and maturing security initiatives—governance-based groups, for example—to inspect and update this implicit policy on a firm-wide basis.

Choose Application Controls

Engineering-driven security cultures naturally favor security controls they can apply to software directly in the form of features [SFD1.1]. This is unsurprising, as delivering security features has all the hallmarks of such a culture's objectives: delivering subject-matter experience as software, impacting the critical path of delivery, and accelerating that delivery. Depending on the way an organization delivers to its customers, application controls can take the form of microservices (e.g., authentication or other identity and access management), 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).

Defensively, engineering-led security groups have taken steps to tackle the prevention of certain classes of vulnerability in a wholesale manner [CMVM3.1], using development frameworks that obviate them, an effort we've seen decrease in governance-led organizations. Security engineers in these groups are often asked their opinion about framework choices and are often empowered to incorporate their understanding of security features and security posture tradeoffs as part of the selection and implementation process. As part of the critical path to software delivery, these engineers can then tune the framework's implementation to the team's and organization's specific situation.

MATURING ENGINEERING-LED EFFORTS

As the foundations of an engineering-led security effort become more concrete, its leaders seek to deepen the technical controls applied, apply all controls to a broader base of the organization's software portfolio and infrastructure, and generally scale their efforts.

Because engineering-led security culture relies heavily on the individual contributions of security engineers distributed within development teams, these firms seek to follow through on what these dispersed engineers have started. Whereas an emerging practice might have focused on automation to ensure host and network security basics, they will now also undertake and incrementally improve vulnerability discovery. They will continue to broaden the catalog of security features delivered by security engineers to meet their view of security, usually as aligned with quality and resiliency rather than centralized corporate governance. Rather than creating new policy, for example, engineering-led groups might formalize the incident response experience accumulated to date into optimized internal process and use code snippets to communicate incident feedback to development teams.

In addition to incremental progress on the activities that security engineers have begun to define, engineering-led security efforts will also seek to apply to the organization as a whole what security engineers have delivered to one development team. This means documenting and sharing software process, extracting explicit organizational policies and standards from existing automation, and formalizing identification of data-driven obligations such as those due to PCI or to other PII use.

Upgrade Incident Response

Governance-based and engineering-led groups alike conduct incident response. Engineering-led teams leverage DevOps engineers to help make connections between those events and alerts raised in production and the artifacts, pipelines, repositories, and teams responsible [CMVM1.1]. This crucial traceability mechanism allows these groups to effectively prioritize security issues on which the security initiative will focus. Feedback from the field essentially replaces the top N lists governance-led organizations use to establish priorities.

Security engineers who are in development teams and are more familiar with application logic might be able to facilitate more 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 more tailored response [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 source for these individuals. Under these circumstances, incident response moves at the speed of a well-practiced single team rather than that of an interdepartmental playbook.

“Because engineering-led security culture relies heavily on the individual contributions of security engineers distributed within development teams, these firms seek to follow through on what these dispersed engineers have started.”

Do Defect Discovery

As firms mature, we see evidence of them building out scalable defect discovery practices. Specifically, these cultures seek to provide highly actionable information about potential security issues to developers proactively. However, visibility into potential vulnerability must come without disrupting CI/CD pipelines with tools that have long execution times, without generating large volumes of perceived false positives, and without impeding delivery velocity (e.g., through broken builds or inadmissible promotion) except under exceptionally clear and convincing circumstances.

Our observation is that engineering-led groups build discovery capability incrementally, with security engineers prototyping new detective capability shoulder-to-shoulder with development teams. Prototyping likely includes in-band triage and developer buy-in to the test findings, their accuracy, and their importance, as well as suggested remediation.

Because they're often free, easy to find, and easy to use, engineering-led cultures tend to start with open source or "freemium" security tools to accomplish various in-band automated code review and testing activities. These tools are often smaller and run quickly due to a limited focus. After these tools are integrated within a pipeline and producing baseline measurements, other stakeholders such as champions might augment the tools and rulesets to, for example, scan for adherence to internal secure coding standards on an incremental basis and across more of the portfolio. The organization's larger, commercial security testing tools that execute over extended periods typically continue to get used at checkpoints relevant to governance groups, with those tests usually being out-of-band of CI/CD pipelines due to the time taken to execute and the friction introduced.

Engineering-Led Heuristics

Our observations are that engineering-led groups are starting with open source and home-grown security tools, with much less reliance on "big box" vulnerability discovery products. Generally, these groups hold to two heuristics:

- **Lengthening time to (or outright preventing) delivery is unacceptable. Instead, organize to provide telemetry and then respond asynchronously through virtual patching, rollback, or other compensating controls.**
- **Build vulnerability detective capability incrementally, in line with a growing understanding of software misuse and abuse and associated business risk, rather than purchasing boxed security standards delivered as part of a vendor's core ruleset in a given tool.**

These groups might build on top of in-place test scaffolding, might purposefully extend open source scanners that integrate cleanly with their development toolchain, or both. Extension often focuses on a different set of issues than characterized in general lists such as the OWASP Top 10 or even the broader set of vulnerabilities found by commercial tools. Instead, these groups might focus on issues such as denial of service, misuse/abuse of business functionality, or enforcement of the organization's technology-specific coding standards (even when these are implicit rather than written down) as defects to be discovered and remediated.

Document the SSDL

Engineering-led cultures typically eschew document- or presentation-based deliverables in favor of code-based deliverables. As the group seeks to apply its efforts to a larger percentage of the firm's software, it might work to institute some form of knowledge-sharing process to get the security activities applied across teams. To this end, security leaders might create a "one-pager" describing the security tools and techniques to be applied throughout software's lifecycle [SM1.1], make that resource available through organizational knowledge management, and evangelize it through internal knowledge-sharing forums.

"As the group seeks to apply its efforts to a larger percentage of the firm's software, it might work to institute some form of knowledge-sharing process to get the security activities applied across teams."

Unlike many governance-driven groups, we have found that engineering-led groups explicitly and purposefully incentivize security engineers to talk externally about those security tools they've created (or customized), such as at regional meet-ups and conferences. Security leads might use these incentives and even open source projects as a way to invite external critique and ensure frequent maintenance and improvement on the tools and frameworks their engineers create, without continuing to tie up 100% of that engineer's bandwidth indefinitely.

SSDL documentation might be made available through an internal or even external source code repository, along with other related material that aids uptake and implementation by development teams. A seemingly simple step, this makes it very easy for development teams to conform to the SSDL within their existing toolchains and cultural norms. It seems especially useful for onboarding new team members by decreasing the time to productivity.

OPTIMIZING ENGINEERING-LED EFFORTS

To some extent, engineering-led groups take an optimizing approach from the beginning. Security efforts are built on contributions of engineers, delivering software early and often, and constantly improving rather than relying on explicit strategy, backed by policies, built top-down, and pushed everywhere through organizational mandate over time.

It's clear that some of these groups have achieved an "optimizing" state of maturity for some of their software security capabilities. However, the BSIMM does not yet contain enough data to generalize about this type of effort in terms of which activities such groups are likely to conduct or how implementation of those activities might differ from their form in governance-driven groups. We will continue to track engineering-led groups and look for patterns and generalizations in the data that might give us insight into how they achieve some maturity that aligns with expectations across the organization and the software portfolio. It might be the case that over the next few years, we will once again see a merging of engineering-led efforts and governance-led efforts into a single, firm-wide SSI.

PART TWO: BSIMM11

The BSIMM is now in its eleventh iteration. In this section, we talk about its history and the underlying model, as well as the changes we made for BSIMM11. We describe the roles we typically see in an SSI and some related terminology. We conclude this section with guidance on how to use the BSIMM to start, mature, and measure your own SSI.

THE BSIMM11 FRAMEWORK

The BSIMM is organized as a set of 121 activities in an SSF, represented here. The framework includes 12 practices that are organized into four domains.

DOMAINS			
GOVERNANCE	INTELLIGENCE	SSDL TOUCHPOINTS	DEPLOYMENT
Practices that help organize, manage, and measure a software security initiative. Staff development is also a central governance practice.	Practices that result in collections of corporate knowledge used in carrying out software security activities throughout the organization. Collections include both proactive security guidance and organizational threat modeling.	Practices associated with analysis and assurance of particular software development artifacts and processes. All software security methodologies include these practices.	Practices that interface with traditional network security and software maintenance organizations. Software configuration, maintenance, and other environment issues have direct impact on software security.

PRACTICES			
GOVERNANCE	INTELLIGENCE	SSDL TOUCHPOINTS	DEPLOYMENT
1. Strategy & Metrics (SM) 2. Compliance & Policy (CP) 3. Training (T)	4. Attack Models (AM) 5. Security Features & Design (SFD) 6. Standards & Requirements (SR)	7. Architecture Analysis (AA) 8. Code Review (CR) 9. Security Testing (ST)	10. Penetration Testing (PT) 11. Software Environment (SE) 12. Configuration Management & Vulnerability Management (CMVM)

THE BSIMM11 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 below, organized by practices and levels. More complete descriptions of the activities and examples are available in Part Three of this document.

GOVERNANCE

STRATEGY & METRICS (SM)	COMPLIANCE & POLICY (CP)	TRAINING (T)
LEVEL 1 • SM1.1 Publish process and evolve as necessary. • SM1.2 Create evangelism role and perform internal marketing. • SM1.3 Educate executives. • SM1.4 Implement lifecycle governance.	**LEVEL 1** • CP1.1 Unify regulatory pressures. • CP1.2 Identify PII obligations. • CP1.3 Create policy.	**LEVEL 1** • T1.1 Conduct awareness training. • T1.5 Deliver role-specific advanced curriculum. • T1.7 Deliver on-demand individual training. • T1.8 Include security resources in onboarding.
LEVEL 2 • SM2.1 Publish data about software security internally. • SM2.2 Verify release conditions with measurements and track exceptions. • SM2.3 Create or grow a satellite. • SM2.6 Require security sign-off.	**LEVEL 2** • CP2.1 Build PII inventory. • CP2.2 Require security sign-off for compliance-related risk. • CP2.3 Implement and track controls for compliance. • CP2.4 Include software security SLAs in all vendor contracts. • CP2.5 Ensure executive awareness of compliance and privacy obligations.	**LEVEL 2** • T2.5 Enhance satellite through training and events. • T2.8 Create and use material specific to company history.
LEVEL 3 • SM3.1 Use a software asset tracking application with portfolio view. • SM3.2 Run an external marketing program. • SM3.3 Identify metrics and use them to drive resourcing. • SM3.4 Integrate software-defined lifecycle governance.	**LEVEL 3** • CP3.1 Create a regulator compliance story. • CP3.2 Impose policy on vendors. • CP3.3 Drive feedback from software lifecycle data back to policy.	**LEVEL 3** • T3.1 Reward progression through curriculum. • T3.2 Provide training for vendors and outsourced workers. • T3.3 Host software security events. • T3.4 Require an annual refresher. • T3.5 Establish SSG office hours. • T3.6 Identify new satellite members through observation.

INTELLIGENCE

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

SSDL TOUCHPOINTS

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

DEPLOYMENT

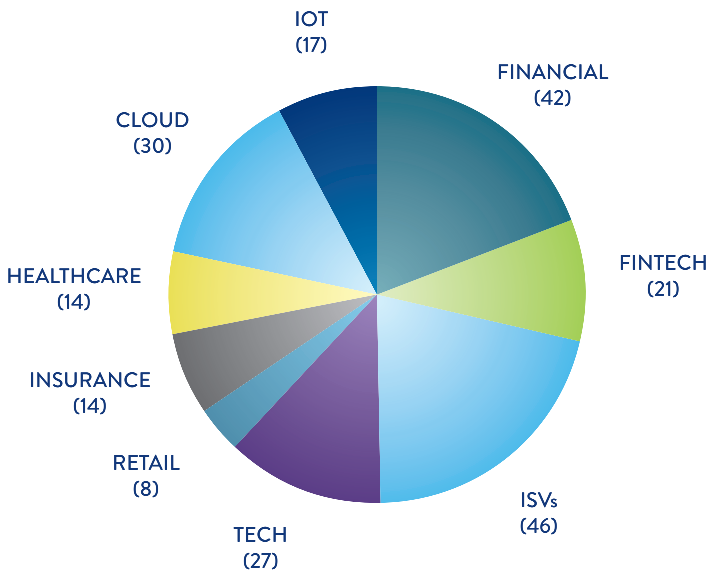
PENETRATION TESTING (PT)	SOFTWARE ENVIRONMENT (SE)	CONFIGURATION MANAGEMENT & VULNERABILITY MANAGEMENT (CMVM)
LEVEL 1 • PT1.1 Use external penetration testers to find problems. • PT1.2 Feed results to the defect management and mitigation system. • PT1.3 Use penetration testing tools internally.	**LEVEL 1** • SE1.1 Use application input monitoring. • SE1.2 Ensure host and network security basics are in place.	**LEVEL 1** • CMVM1.1 Create or interface with incident response. • CMVM1.2 Identify software defects found in operations monitoring and feed them back to development.
LEVEL 2 • PT2.2 Penetration testers use all available information. • PT2.3 Schedule periodic penetration tests for application coverage.	**LEVEL 2** • SE2.2 Define secure deployment parameters and configurations. • SE2.4 Protect code integrity. • SE2.5 Use application containers. • SE2.6 Ensure cloud security basics.	**LEVEL 2** • CMVM2.1 Have emergency response. • CMVM2.2 Track software bugs found in operations through the fix process. • CMVM2.3 Develop an operations inventory of software delivery value streams.
LEVEL 3 • PT3.1 Use external penetration testers to perform deep-dive analysis. • PT3.2 Customize penetration testing tools.	**LEVEL 3** • SE3.2 Use code protection. • SE3.3 Use application behavior monitoring and diagnostics. • SE3.5 Use orchestration for containers and virtualized environments. • SE3.6 Enhance application inventory with operations bill of materials.	**LEVEL 3** • CMVM3.1 Fix all occurrences of software bugs found in operations. • CMVM3.2 Enhance the SSDL to prevent software bugs found in operations. • CMVM3.3 Simulate software crises. • CMVM3.4 Operate a bug bounty program. • CMVM3.5 Automate verification of operational infrastructure security. • CMVM3.6 Publish risk data for deployable artifacts.

WHAT BSIMM11 TELLS US

This section provides information about BSIMM11 participants. We show the current data pool composition, the BSIMM11 Scorecard for the 130 firms across 121 activities, and some analysis of the industry verticals represented. We also discuss emerging trends in the BSIMM11 data.

The BSIMM data pool includes participants from a variety of vertical markets. Figure 3 provides counts for those vertical markets with sufficient members to allow reporting on that group. To help protect member privacy, we don't publish details on verticals with very few members.

The BSIMM data yield very interesting analytical results as shown throughout this document. Shown in Table 4 are the highest-resolution BSIMM data that are published. Organizations can use these data to note how often we observe each activity across all 130 participants and use that information to help plan the next areas of focus. Activities that are broadly popular across all vertical markets will likely benefit your organization as well.



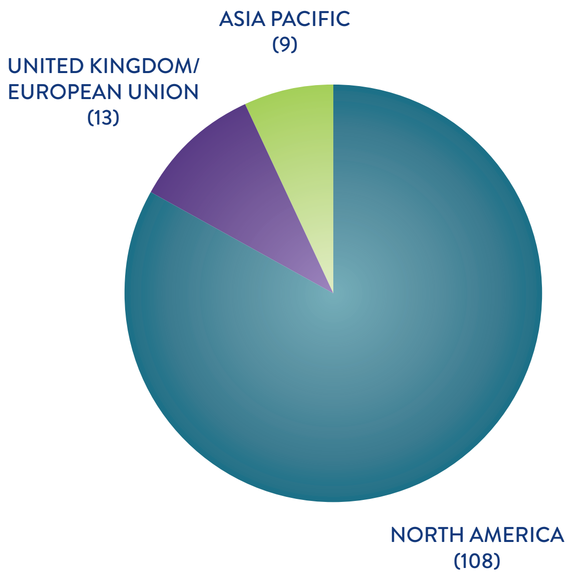


Figure 3. BSIMM11 Participating Firms. *Participant counts per tracked vertical in the BSIMM11 data pool. Note that some firms are in multiple vertical markets and some firms are in verticals not listed here, such as energy and telecoms.*

GOVERNANCE			INTELLIGENCE			SSDL TOUCHPOINTS			DEPLOYMENT		
ACTIVITY	BSIMM11 FIRMS (out of 130)	BSIMM11 FIRMS (%)	ACTIVITY	BSIMM11 FIRMS (out of 130)	BSIMM11 FIRMS (%)	ACTIVITY	BSIMM11 FIRMS (out of 130)	BSIMM11 FIRMS (%)	ACTIVITY	BSIMM11 FIRMS (out of 130)	BSIMM11 FIRMS (%)
STRATEGY & METRICS			ATTACK MODELS			ARCHITECTURE ANALYSIS			PENETRATION TESTING		
[SM1.1]	94	72.3%	[AM1.2]	81	62.3%	[AA1.1]	114	87.7%	[PT1.1]	114	87.7%
[SM1.2]	70	53.8%	[AM1.3]	38	29.2%	[AA1.2]	41	31.5%	[PT1.2]	100	76.9%
[SM1.3]	75	57.7%	[AM1.5]	57	43.8%	[AA1.3]	32	24.6%	[PT1.3]	89	68.5%
[SM1.4]	117	90.0%	[AM2.1]	12	9.2%	[AA1.4]	67	51.5%	[PT2.2]	30	23.1%
[SM2.1]	64	49.2%	[AM2.2]	10	7.7%	[AA2.1]	23	17.7%	[PT2.3]	29	22.3%
[SM2.2]	61	46.9%	[AM2.5]	16	12.3%	[AA2.2]	24	18.5%	[PT3.1]	21	16.2%
[SM2.3]	55	42.3%	[AM2.6]	10	7.7%	[AA3.1]	11	8.5%	[PT3.2]	10	7.7%
[SM2.6]	65	50.0%	[AM2.7]	14	10.8%	[AA3.2]	1	0.8%			
[SM3.1]	27	20.8%	[AM3.1]	3	2.3%	[AA3.3]	6	4.6%			
[SM3.2]	4	3.1%	[AM3.2]	4	3.1%						
[SM3.3]	18	13.8%	[AM3.3]	4	3.1%						
[SM3.4]	1	0.8%									
COMPLIANCE & POLICY			SECURITY FEATURES & DESIGN			CODE REVIEW			SOFTWARE ENVIRONMENT		
[CP1.1]	94	72.3%	[SFD1.1]	102	78.5%	[CR1.2]	79	60.8%	[SE1.1]	73	56.2%
[CP1.2]	112	86.2%	[SFD1.2]	76	58.5%	[CR1.4]	100	76.9%	[SE1.2]	121	93.1%
[CP1.3]	86	66.2%	[SFD2.1]	32	24.6%	[CR1.5]	49	37.7%	[SE2.2]	39	30.0%
[CP2.1]	55	42.3%	[SFD2.2]	51	39.2%	[CR1.6]	41	31.5%	[SE2.4]	33	25.4%
[CP2.2]	47	36.2%	[SFD3.1]	14	10.8%	[CR1.7]	50	38.5%	[SE2.5]	31	23.8%
[CP2.3]	61	46.9%	[SFD3.2]	14	10.8%	[CR2.6]	23	17.7%	[SE2.6]	36	27.7%
[CP2.4]	48	36.9%	[SFD3.3]	4	3.1%	[CR2.7]	24	18.5%	[SE3.2]	13	10.0%
[CP2.5]	70	53.8%				[CR3.2]	7	5.4%	[SE3.3]	7	5.4%
[CP3.1]	26	20.0%				[CR3.3]	3	2.3%	[SE3.5]	22	16.9%
[CP3.2]	16	12.3%				[CR3.4]	2	1.5%	[SE3.6]	12	9.2%
[CP3.3]	8	6.2%				[CR3.5]	1	0.8%			
TRAINING			STANDARDS & REQUIREMENTS			SECURITY TESTING			CONFIG. MGMT. & VULN. MGMT.		
[T1.1]	83	63.8%	[SR1.1]	93	71.5%	[ST1.1]	104	80.0%	[CMVM1.1]	108	83.1%
[T1.5]	38	29.2%	[SR1.2]	90	69.2%	[ST1.3]	89	68.5%	[CMVM1.2]	102	78.5%
[T1.7]	53	40.8%	[SR1.3]	94	72.3%	[ST2.1]	42	32.3%	[CMVM2.1]	94	72.3%
[T1.8]	41	31.5%	[SR2.2]	64	49.2%	[ST2.4]	20	15.4%	[CMVM2.2]	97	74.6%
[T2.5]	32	24.6%	[SR2.4]	60	46.2%	[ST2.5]	16	12.3%	[CMVM2.3]	65	50.0%
[T2.8]	28	21.5%	[SR2.5]	44	33.8%	[ST2.6]	13	10.0%	[CMVM3.1]	5	3.8%
[T3.1]	4	3.1%	[SR3.1]	30	23.1%	[ST3.3]	5	3.8%	[CMVM3.2]	11	8.5%
[T3.2]	22	16.9%	[SR3.2]	9	6.9%	[ST3.4]	2	1.5%	[CMVM3.3]	13	10.0%
[T3.3]	16	12.3%	[SR3.3]	9	6.9%	[ST3.5]	2	1.5%	[CMVM3.4]	16	12.3%
[T3.4]	19	14.6%	[SR3.4]	25	19.2%	[ST3.6]	0	0.0	[CMVM3.5]	8	6.2%
[T3.5]	7	5.4%							[CMVM3.6]	0	0.0
[T3.6]	1	0.8%									

Table 4. BSIMM11 Scorecard. The scorecard shows how often each of the activities in the BSIMM were observed in the BSIMM11 data pool from 130 firms. Note that the data fall naturally into levels by observation count.

In the BSIMM11 Scorecard, we also identified the most common activity in each practice (highlighted in gray in the scorecard above). These 12 activities were observed in at least 81 (62%) of the 130 firms we studied (see Part Three for "Table 5. Most Common Activities Per Practice").

To provide another view into this table data, we created spider charts by noting the highest-level activity observed for each practice per BSIMM participant (a "high-water mark") and then averaging these values over the group of 130 firms to produce 12 numbers (one for each practice). The resulting spider chart (Figure 4) plots these values on 12 spokes corresponding to the 12 practices. Note that performing level 3 (the outside edge) activities is often a sign of SSI maturity but only because organizations tend to start with common activities (level 1) and build from there toward uncommon activities. Other interesting analyses are possible, of course. For example, see https://ieeexplore.ieee.org/document/8409917.

By computing these high-water mark values and an observed score for each firm in the study, we can also compare relative and average maturity for one firm against the others. The range of observed scores in the current data pool is 5 for the lower score and 83 for the higher score, indicating a wide range of SSI maturity levels in the BSIMM11 data.

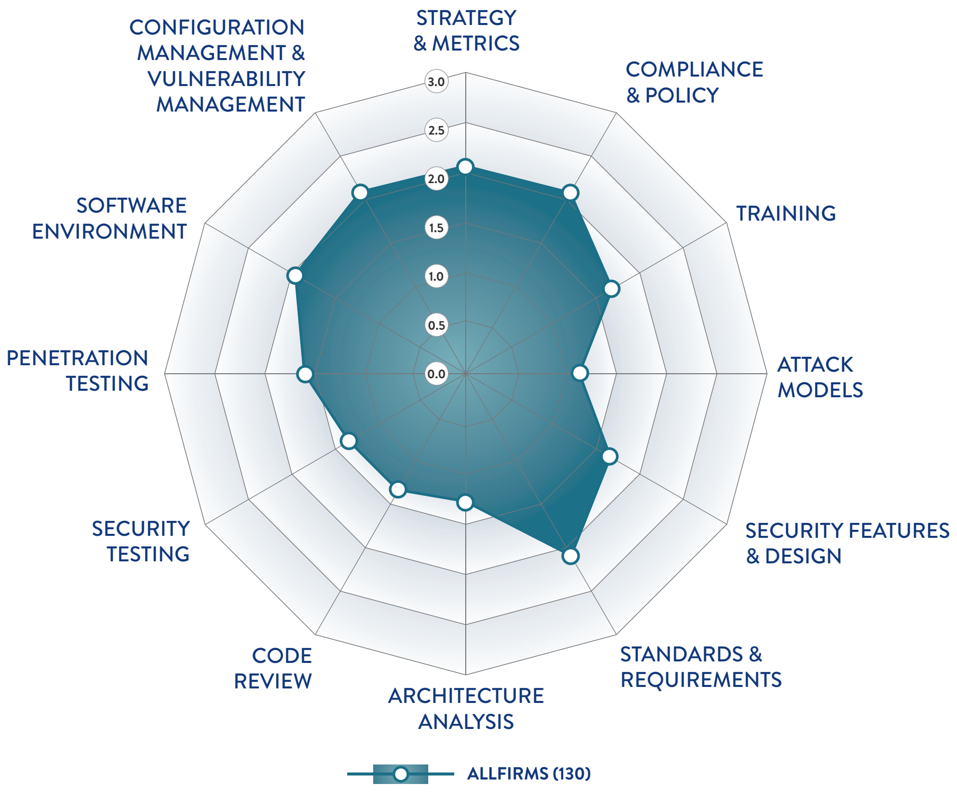


Figure 4. AllFirms Spider Chart. *This diagram shows the average of the high-water mark collectively reached in each practice by the 130 BSIMM11 firms. Collectively across these firms, we observed more level 2 and level 3 activities in practices such as Strategy & Metrics and Compliance & Policy compared to practices such as Attack Models and Architecture Analysis.*

We're pleased that the BSIMM study continues to grow year after year. The dataset we report on here is nearly 40 times the size it was for the original publication. Note that once we exceeded a sample size of 30 firms, we began to apply statistical analysis, yielding statistically significant results.

BSIMM11 AND INDUSTRY VERTICALS ANALYSIS

Spider charts are also useful for comparing groups of firms from particular industry verticals. The following figures show data from verticals in BSIMM11.

Cloud, Internet of Things, and high-technology firms are three of the most mature verticals in the BSIMM11 data pool (see Figure 5). On average, cloud firms (which are not necessarily equivalent to cloud service providers) are somewhat more mature in the Governance and Intelligence domains compared to the technology and Internet of Things firms.

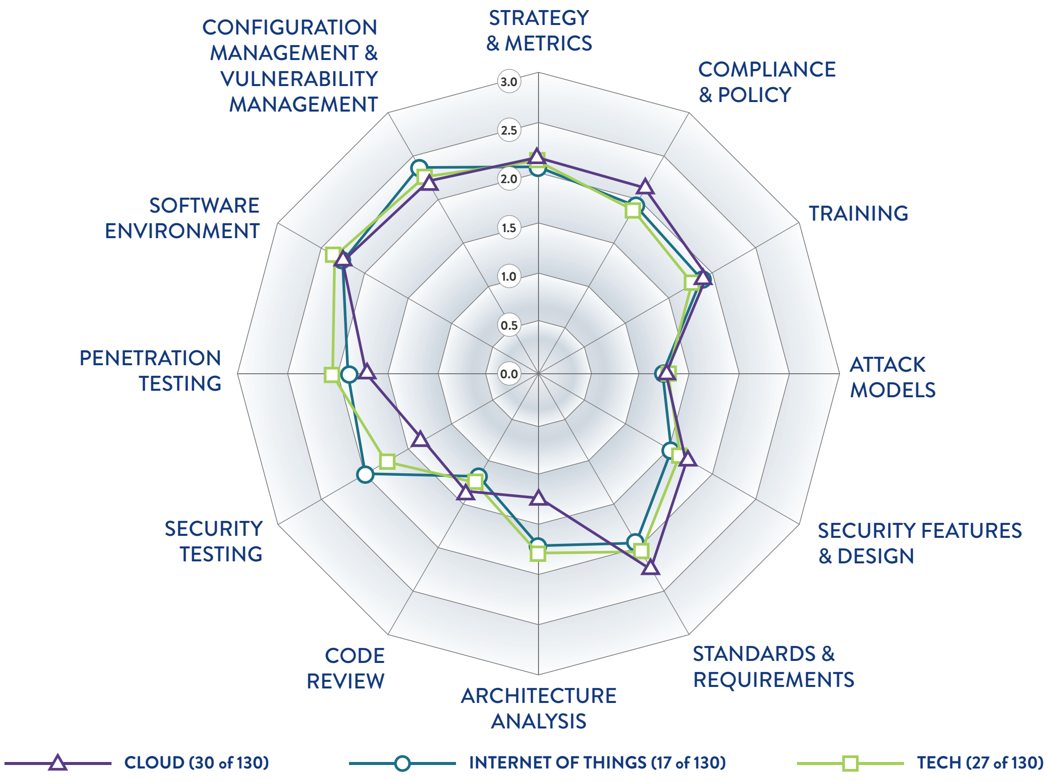


Figure 5. Cloud vs. Internet of Things vs. Tech Spider Chart. Mature verticals still show distinct differences in the Governance and Intelligence domains and particularly in the Architecture Analysis, Security Testing, and Penetration Testing practices.

By the same measure, technology and Internet of Things firms show greater maturity in the SSDL Touchpoints and Deployment domains, especially in the Architecture Analysis, Security Testing, and Penetration Testing practices. Despite these obvious differences, there is a great deal of overlap. We believe that technology stacks and architectures, and therefore many of the associated software security activities, between these three verticals are continuing to converge.

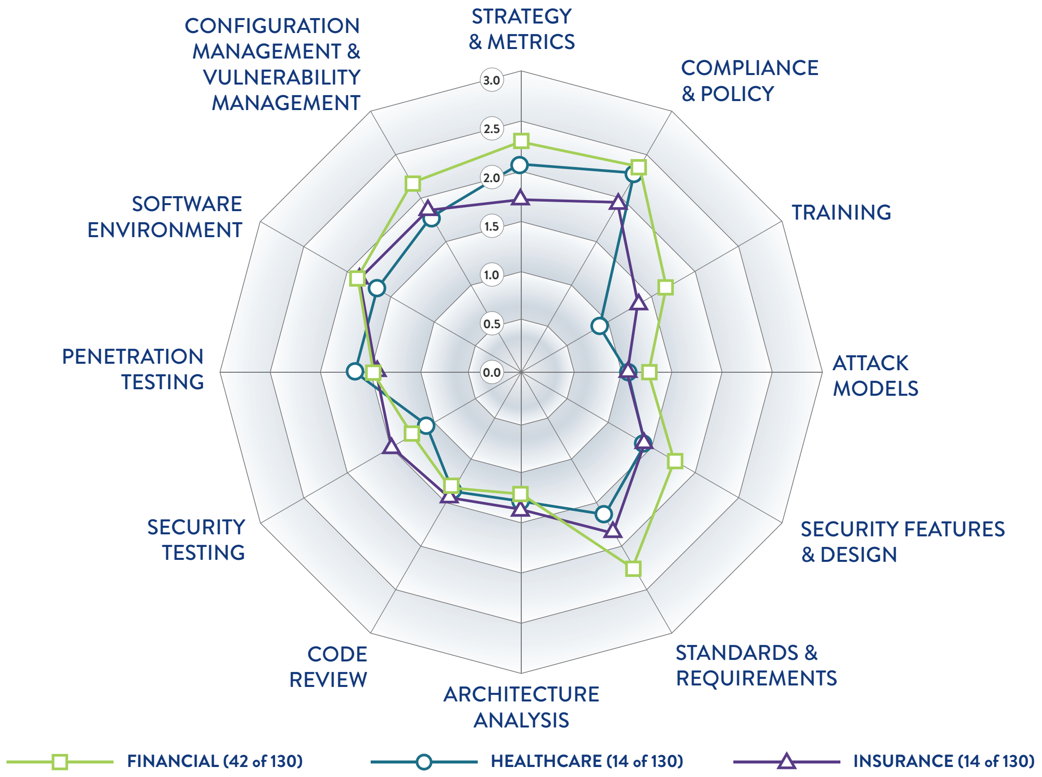


Figure 6. Financial vs. Healthcare vs. Insurance Spider Chart. *Although they share similar compliance drivers, these groups of organizations have different average maturity levels.*

Three verticals in the BSIMM operate in highly regulated industries: financial services, healthcare, and insurance (see Figure 6). In our experience with the BSIMM, large financial services firms reacted to regulatory changes and started their SSIs much earlier than insurance and healthcare firms. Even as the number of financial services firms has grown significantly over the past five years, with a large influx into the BSIMM data pool of newly started initiatives, the financial services SSG average age at last assessment time is 4.9 years versus 3.8 years for insurance and 3.7 years for healthcare. Time spent by financial services firms maturing their collective SSIs shows up clearly in the side-by-side comparison. Although organizations in the insurance vertical include some mature outliers, the data for these three regulated verticals show insurance lags behind in the Strategy & Metrics, Compliance & Policy, and Attack Models practices, while moving above average in the Security Testing practice. Compared to financial services firms, we see a similar contrast in healthcare, which achieves par in Compliance & Policy, Architecture Analysis, Code Review, and Penetration Testing, but lags in other practices.

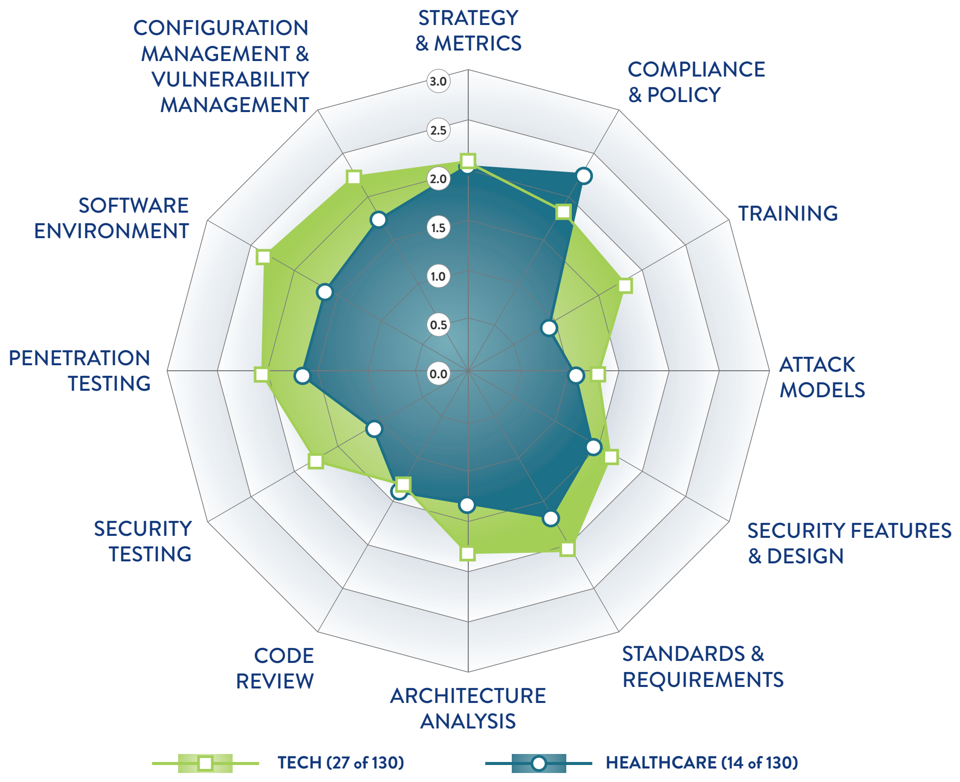


Figure 7. Tech vs. Healthcare Spider Chart. *Although healthcare firms are increasingly building devices and associated services, their overall maturity lags behind technology firms that do similar things.*

In the BSIMM population, we can find large gaps between the maturity of verticals, even when the technology stacks might be similar. Consider Figure 7, which directly compares the current technology and healthcare verticals. In this case, there is an obvious delta between technology firms that build devices tied to back-end services and healthcare firms that increasingly build devices tied to back-end services. The disparity in maturity extends to most practices, although the healthcare vertical is predictably ahead in the Compliance & Policy practice. Fortunately for organizations that find themselves behind the curve, the experiences of many BSIMM participants provide a good roadmap to faster maturity.

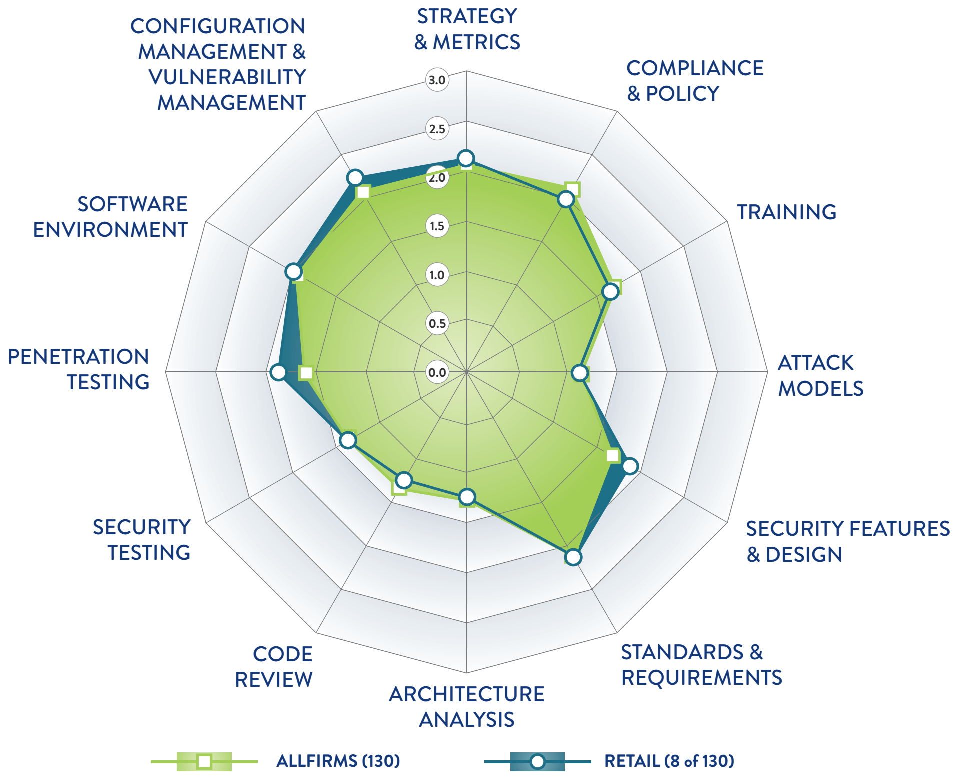


Figure 8. AllFirms vs. Retail Spider Chart. While it might have taken some years for retail firms to begin using the BSIMM in earnest, they have been working on their SSIs.

For the third year, the BSIMM presents data on the retail vertical. This group, with an average SSG age at time of last assessment of 4.2 years and average SSG size of 8.6 full-time people, seems to track closely to the overall data pool (Figure 8). The most obvious differences are in the Security Features & Design and Penetration Testing practices, where retail participants are somewhat ahead of the average for all firms.

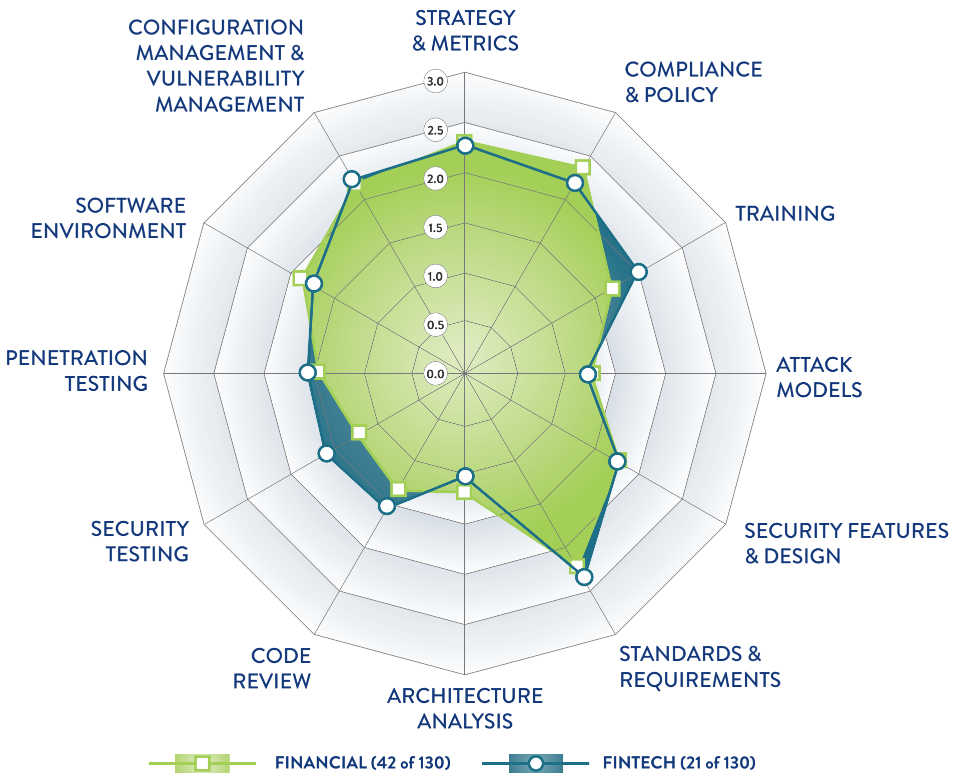


Figure 9. Financial vs. FinTech Spider Chart. *The financial services firms and financial technology firms that produce software for financial services firms show a few distinct differences.*

For the first time, the BSIMM presents data on the FinTech vertical, the firms that are effectively ISVs specifically for financial services software. We carefully reviewed the growing pool of firms in the financial vertical and moved some to the FinTech vertical, then also added new firms to the appropriate financial or FinTech vertical, or both when appropriate. As Figure 9 shows, the two verticals track fairly closely with each other, with a primary difference being in the Training practice and reasonably predictable differences in the Code Review and Security Testing practices.

EMERGING TRENDS IN BSIMM11

Several trends observed in BSIMM10 have made their mark on the BSIMM11 data. Activity trends in DevOps groups continue to be observed more often and the activity implementation is increasingly automated.
Trends in SSI maturity continue to be molded by the data pool constituency.

ACTIVITY TRENDS

The BSIMM10 data revealed noteworthy trends in the security efforts led by development, security, and operations groups in building, deploying, and operating software. We observed both new activities and changes to implementation approaches for existing activities in a sufficiently consistent and comprehensive manner to reflect what we call engineering-led security initiatives, where SSGs might structurally manifest within engineering groups. To the extent that a corporate SSG exists separate from engineering or product security, the BSIMM11 data indicate a continued trend of shifting away from "mandating behavior and auditing compliance" to lending resources or staff to DevOps practices with an explicit objective to directly contribute security efforts to the critical path for software delivery value streams.

Increased reliance on engineering-led activities is a theme the reader will notice throughout the BSIMM11 study. This section outlines some related trends, characterizing engineering-led activities and other observed themes among the data. Whereas the BSIMM10 data showed engineering-led experimentation beginning to succeed and endure, the BSIMM11 data reflect evidence that some firms have transcended experimentation; these firms are now choosing to rely on engineering-led ideals for implementing software security capabilities. We've attempted to characterize trends we are now observing in what makes engineering-led initiatives different in terms of activities, as well as what corporate security practitioners and executives must do to earn a seat at the table, ensuring security is part of business delivery value streams.

Activity Trend: Governance as Code

The BSIMM10 data showed evidence that organizations have begun successfully replacing manual, human-driven governance activities with automation. To date, the BSIMM has differentiated between having written, human-readable security policy versus configuration or code that verifies adherence to security policy. Increasingly in BSIMM11, especially in engineering-led security initiatives, the sole and ultimate source of security standards and policy might be human-readable configuration or simplified code that conducts vulnerability discovery, hardens artifacts (e.g., containers) for secure deployment or operation, or verifies container and cloud configuration at runtime—the essence of software-defined lifecycle governance.

For it to be an effective part of a security initiative, security policy must remain human-readable. That is, humans must be able to easily access, read, and audit against this policy without being code literate. In fast-paced delivery environments, engineering and operations groups not only make self-service modifications to the corporate, top-down policy as they respond to operational issues (e.g., defects, insecure configurations) but also create new policy along the way. While individual teams might evolve policy for local needs, the SSI must provide a way of incorporating automated detection of drift in this policy beyond the mandated guardrails, often as code configurations without human audit.

"For it to be an effective part of a security initiative, security policy must remain human-readable."

Activity Trend: Continuous Defect Discovery

Long-standing development trends such as continuous integration and testing have motivated changes in defect discovery tools and usage. The concept of a synchronous governance checkpoint (or gate) relying on data from a point-in-time scan has become an unacceptable fit with continuous delivery culture. As a result, developers and satellite members pushed for "faster scan times" from security tools, so that these tools would fit within a delivery pipeline's cadence without causing undue friction. Though their architecture and implementation approaches vary widely, we're now observing firms that are implementing modern defect discovery tools (both open source and commercial) and preferring monitoring and continuous reporting approaches, which extends to configuration, deployment, and usage issues, rather than simple defect discovery. When information from code inspection, observation of usage, or a combination of the two exceed a threshold, these tools emit a security issue to be routed through defect management and remediation workflows.

Conceptually, this marks a shift in defect discovery from slowing down development in hopes of deploying more software with fewer vulnerabilities to a realization that there will always be more vulnerabilities, so we can't continuously disturb delivery cadence. The focus has become creating resiliency through an extremely low latency and continuous detect-plan-respond loop. We'll discuss resiliency more in a later trend.

Practically, this means SSG leaders need to resolve something of a mismatch in cadence. On one hand, there is the continuous delivery of individual security events from potentially many security sensors deployed as agents within their development and production environments. On the other is the need to provide trending and reporting on fixed periodic intervals to leadership. Various roles also need to make governance decisions, such as whether to promote to production at key points within a software lifecycle based on all available data gathered along the way. Minute-by-minute metrics for decisions made monthly are probably no more useful than month-to-month metrics for decisions made every minute.

Because so much code lives forever and the ability to find defects in both new and production code improves over time, continuous defect discovery might naturally evolve to include production environments (shift right), emphasizing resilience over time, rather than discovering security defects piecemeal.

Activity Trend: Continuous Activity

Trends in programming languages, container orchestration, microservice orientation, and cloud configuration management have enabled engineering self-service. This has resulted in a "shift everywhere" approach to security activities: some organizations no longer conduct all the steps of traditional security activities in one lifecycle phase, opting to move to continuous efforts spread through many phases. They decompose monolithic efforts (e.g., three-day threat modeling) and then incrementally execute the step-wise process in each phase where the knowledge can be effectively applied to a value stream artifact, with automation whenever possible. The imperative to do things as early as possible remains, with the growing realization that sometimes deployment orchestration or the post-deployment environment reflects the earliest, best opportunity.

In the context of the software development and operational lifecycle, this trend extends beyond defect discovery, as discussed in the previous section, to other activities including configuration governance of containers, APIs, and clouds, and even quality measurement. When conducting activities continuously, security telemetry from SSDL activities is passed from one lifecycle phase to the next, just as software artifacts pass functionally. After deployment, the previous lifecycle's security telemetry re-enters the lifecycle at its beginning, again just like the previous version of software does. Through the implementation of software-defined mechanisms, there is a continuous approach toward governance-as-code.

Conceptually, we're observing the implementation of specific SSDL activities manifesting in a continuous fashion as follows:

- Satellite members, or automation on behalf of the satellite or security champions, might conduct the steps necessary to fulfill an aspect of a particular security activity throughout each and every phase of the lifecycle. In this model, making security decisions and applying governance based on these incremental efforts entails using whatever data are collectively available at that stage—both data that were just collected and data that are available from the previous trip through the lifecycle.
- In an agile world, secure design and other architecture activities have migrated from the initiation phase of a lifecycle and spread throughout lifecycle phases as incremental steps. As examples, security champions might indicate key misuse or abuse cases when documenting user stories and epics, when doing just-in-time secure design as part of ticket grooming, and when doing detailed secure design as part of ticketing and development workflows. Because infrastructure is increasingly codified (i.e., infrastructure-as-code), secure design extends to network configuration, identity and access management, virtualization, and orchestration. All engineering work—both application and infrastructure features and defects—is tracked on the backlog.
- As a simplification, the application granularity was the only artifact where an organization applied code review and usually as a point-in-time event (e.g., a gate). The decade-old trend to shift left to the moment defects are introduced to a codebase, such as in a pull request, still remains. At the same time, however, because more infrastructure and orchestration are code, and because programming languages and developer frameworks are becoming more dynamic (potentially instantiating even more code at runtime), there's a shift right in code review to provide visibility into security risks across the entire stack and software lifecycle. In some organizations, post-deployment analysis of configuration and binaries occurs to verify adherence to coding standards and to catch configuration drift. Given the application and all of its adjacencies (e.g., container configuration, orchestration code), code review as an activity then, can be said to occur not just at a point in time (historically, at a gate), but all the way from ideation through deployment and operations.

With continuous activity comes an important challenge for firms. SSG leaders are largely left to their own devices, tasked with finding a way to coalesce individual security events from myriad sets of agents and activity steps, then create a realistic picture of risk from which they can govern, report, and analyze trends. The current state seems to be a necessary reliance on automation to provide aggregation for increasingly fragmented single data points to make them actionable as vulnerability or risk data.

"SSG leaders are largely left to their own devices, tasked with finding a way to coalesce individual security events from myriad sets of agents and activity steps, then create a realistic picture of risk from which they can govern, report, and analyze trends."

Activity Trend: Security as Resilience and Quality

The BSIMM10 data indicated some success: developers, engineers (e.g., site reliability engineers), and those operating software—indeed, everyone—seem to view some aspect of security as their responsibility. In BSIMM11, contributing to business value streams might mean learning esoteric vulnerability and exploitation detail, combined with integrating and operating myriad sets of tools to successfully implement security at speed and scale. To ensure that everyone contributing to security efforts focuses precious time on SSI priorities, there is a push for many SSDL activities (particularly defect discovery and production monitoring and feedback loop activities) to become business as usual. Naturally, organizations look to optimize these new business-as-usual security capabilities as they would any other set of functional delivery activities.

For some time, organizations have been dramatically improving functional quality assurance practices, as well as nonfunctional aspects of assurance, such as availability or load and performance testing. Adding to quality assurance—the pursuit of building more reliable software proactively by adding activities to the development lifecycle—organizations have added practice areas like A/B testing, chaos engineering, and others in pursuit of resilience, the ability of an organization's systems to carry on despite stresses or attacks. Resilience practices have achieved impressive maturity within some engineering-led organizations.

Many security activities fit naturally within quality assurance practices for engineering-led initiatives, particularly defect discovery activities like SAST, SCA, and even DAST, which has historically resisted automation. Previously observed focus on the use of open source testing tools and automation has been reinforced in the last 12 months of observation.

Software security fits naturally within resilience practices as well, although it might seem unnatural to traditional security leaders. Philosophically, focusing on resilience means conceding that delivery of vulnerable software will occur, and there must be an increased maturity in implementing solutions involving people, process, and technology to assure that once a vulnerability is discovered, software can be put promptly back into a safe (or nonvulnerable) state.

We observe engineering-led security initiatives having built the capability to provide resilience not only in the face of reliability and scalability challenges but also in the face of attack. This requires building capabilities focused on inventorying software, on cataloging its constituency, on understanding the means by which it was built, configured, and deployed, and on the ability to re-deploy based on security telemetry. Organizations that have incorporated security response into resiliency have built both a cultural and technology-based means of quickly convening the right individuals to respond to what's observed in production SIEMs, intrusion prevention systems, threat intelligence feeds, and other security sensors. Having traceability within software delivery pipelines allows these organizations to make some remediation decisions (e.g., patching, rollback, applying a WAF rule, locking down access) automatically. Where automated response isn't possible, these firms strive to quickly put highly specific risk information in decision-makers' hands to facilitate response.

> *"We observe engineering-led security initiatives having built the capability to provide resilience not only in the face of reliability and scalability challenges but also in the face of attack."*

DATA TRENDS

As the BSIMM community grew, we added a greater number of firms with newer SSIs and began to track new verticals that have less software security experience. Thus, we expected to see a direct impact on the data.

Impact on Average Data Pool Score

Adding firms with less experience decreased the average score to 33.1 in BSIMM8 from 33.9 in BSIMM7 and 36.7 in BSIMM6, even as remeasurements have shown that individual firm maturity increases over time. For BSIMM9, however, the average score increased to 34.0, increased again to 35.6 for BSIMM10, and is now at 38.2 (median 37.0) for BSIMM11.

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:

- The average SSG age for new firms entering BSIMM6 was 2.9 years; it was 3.37 years for BSIMM7, 2.83 years for BSIMM8, and increased to 4.57 years for BSIMM9. On the other hand, the average SSG age for new firms in BSIMM10 was 3.42 years and increased to 4.3 years for BSIMM11.
- A second reason appears to be an increase in firms continuing to use the BSIMM to guide their initiatives. BSIMM7 included 11 firms that received their second or higher assessment. That figure increased to 12 firms for BSIMM8, increased to 16 firms for BSIMM9 and BSIMM10, and dropped slightly to 14 firms for BSIMM11.
- A third reason appears to be the effect of firms aging out of the data pool. We removed 72 firms for BSIMM-V through BSIMM10 and an additional 19 firms for BSIMM11; interestingly, 10 of the 91 firms that had once aged out of the BSIMM data pool have subsequently rejoined with a new assessment.

We also see this trend in mature verticals such as financial services, where average overall maturity decreased to 35.6 in BSIMM8 from 36.2 in BSIMM7 and 38.3 in BSIMM6. For BSIMM9, however, the average financial services score increased to 36.8, then to 37.6 for BSIMM10, and 40.0 for BSIMM11. Of potential impact here, five of the 11 firms dropped from BSIMM9 due to data age were in the financial services group, while that figure was two of 17 firms dropped for BSIMM10 and eight of 19 firms for BSIMM11. Also note that we reorganized the financial services group for BSIMM11 while creating the financial technology group.

Impact on Satellite and Activities

Note that a large number of firms with no satellite continue to exist in the community, which causes the median satellite size to be zero (68 of 130 firms had no satellite at the time of their current assessment, and nearly 60% of the firms added for BSIMM11 had no satellite at assessment time). BSIMM participants, however, continue to report that the existence of a satellite is directly tied to SSI maturity. For the 62 BSIMM11 firms with a satellite at assessment time, the average size was 107 with a median of 30. Notably, the average score for the 62 firms with a satellite is 45.2, while the average score for the 68 firms without a satellite is 31.8.

For BSIMM8, we zoomed in on two particular activities as part of our analysis. Observations of [AA3.3 Make the SSG available as an AA resource or mentor] dropped to 2% in the BSIMM8 community from 5% in BSIMM7, 17% in BSIMM6, and 30% in BSIMM-V. However, observations rose to 3% for BSIMM9 and BSIMM, and rose again to 5% for BSIMM11. Observations of [SR3.3 Use secure coding standards] dropped to 14% in BSIMM8 from 18% in BSIMM7, 29% in BSIMM6, and 40% in BSIMM-V. In this case, the slide continued to 8% for BSIMM9 and 7% in BSIMM10 and BSIMM11. This kind of change can be seen in activities spanning all 12 practices. In some cases, it appears that instead of focusing on a robust, multi-activity approach to a given practice, many firms have a tendency to pick one figurehead activity (e.g., static analysis with a tool or penetration testing) on which to focus their investment in money, people, and effort. In other cases, it appears that some SSGs have moved away from being the source of expertise on software security architecture and secure coding standards, without the organization having those skills and knowledge appropriately spread across the product teams.

> *“Firms that have been in the BSIMM community for multiple years have, with one or two exceptions, always increased the number of activities they are able to deploy and maintain over time.”*

Firms that have been in the BSIMM community for multiple years have, with one or two exceptions, always increased the number of activities they are able to deploy and maintain over time. We expect the majority of newer firms entering the BSIMM population to do the same.

In Table 5, we show the most common activities per 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 highly successful initiatives. This suggests that if an organization is working on an initiative of its own, it should consider these 12 activities particularly carefully.

MOST COMMON ACTIVITIES PER PRACTICE	
ACTIVITY	DESCRIPTION
[SM1.4]	Implement lifecycle governance.
[CP1.2]	Identify PII obligations.
[T1.1]	Conduct awareness training.
[AM1.2]	Create a data classification scheme and inventory.
[SFD1.1]	Integrate and deliver security features.
[SR1.3]	Translate compliance constraints to requirements.
[AA1.1]	Perform security feature review.
[CR1.4]	Use automated tools along with manual review.
[ST1.1]	Ensure QA performs edge/boundary value condition testing.
[PT1.1]	Use external penetration testers to find problems.
[SE1.2]	Ensure host and network security basics are in place.
[CMVM1.1]	Create or interface with incident response.

Table 5. Most Common Activities Per Practice. This table shows the most commonly observed activity in each of the 12 BSIMM practices for the entire data pool of 130 firms. This means each activity has broad applicability across a wide variety of SSIs.

Of course, Table 5 (showing the most common activity in each practice) isn't the same as the list of the most common activities (Table 6). If you're working on improving your company's SSI, you should consider these 20 activities as likely important to your plans.

BSIMM11 TOP 20 ACTIVITIES BY OBSERVATION COUNT			
RANK	OBSERVATIONS	ACTIVITY	DESCRIPTION
1	121	[SE1.2]	Ensure host and network security basics are in place.
2	117	[SM1.4]	Implement lifecycle governance.
3	114	[AA1.1]	Perform security feature review.
4	114	[PT1.1]	Use external penetration testers to find problems.
5	112	[CP1.2]	Identify PII obligations.
6	108	[CMVM1.1]	Create or interface with incident response.
7	104	[ST1.1]	Ensure QA performs edge/boundary value condition testing.
8	102	[SFD1.1]	Integrate and deliver security features.
9	102	[CMVM1.2]	Identify software defects found in operations monitoring and feed them back to development.
10	100	[CR1.4]	Use automated tools along with manual review.
11	100	[PT1.2]	Feed results to the defect management and mitigation system.
12	97	[CMVM2.2]	Track software bugs found in operations through the fix process.
13	94	[SM1.1]	Publish process and evolve as necessary.
14	94	[CP1.1]	Unify regulatory pressures.
15	94	[SR1.3]	Translate compliance constraints to requirements.
16	94	[CMVM2.1]	Have emergency response.
17	93	[SR1.1]	Create security standards.
18	90	[SR1.2]	Create a security portal.
19	89	[ST1.3]	Drive tests with security requirements and security features.
20	89	[PT1.3]	Use penetration testing tools internally.

Table 6. Top 20 Activities by Observation Count. *Shown here are the most commonly observed activities in the BSIMM11 data.*

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.

SM LEVEL 1

[SM1.1: 94] Publish process and evolve as necessary.

The process for addressing software security is published and broadcast to all stakeholders so that everyone knows the plan. Goals, roles, responsibilities, and activities are explicitly defined. Most organizations pick an existing methodology, such as the Microsoft SDL or the Synopsys Touchpoints, and then tailor it to meet their needs. Security activities, such as those grouped into an SSDL process, are adapted to software lifecycle processes (e.g., waterfall, agile, CI/CD, DevOps) so activities will evolve with both the organization and the security landscape. In many cases, the process is defined by the SSG and published only internally; it doesn't need to be publicly promoted outside the firm to have the desired impact. In addition to publishing the written process, some firms also encode it into an application lifecycle management (ALM) tool as software-defined workflow.

[SM1.2: 70] Create evangelism role and perform internal marketing.

The SSG builds support for software security throughout the organization through ongoing evangelism. This internal marketing function, often performed by a variety of stakeholder roles, keeps executives and others up to date on the magnitude of the software security problem and the elements of its solution. A scrum master familiar with security, for example, could help teams adopt better software security practices as they transform to an agile methodology. Evangelists can increase understanding and build credibility by giving talks to internal groups (including executives), publishing roadmaps, authoring white papers for internal consumption, or creating a collection of papers, books, and other resources on an internal website and promoting its use.

[SM1.3: 75] Educate executives.

Executives are regularly shown the ways malicious actors attack software and the negative business impacts they can have on the organization. They're also shown what other organizations are doing to mature software security, including how they deal with the risks of adopting emerging engineering methodologies with no oversight. By understanding both the problems and their proper resolutions, executives can support the SSI as a risk management necessity. In its most dangerous form, security education arrives courtesy of malicious hackers or public data exposure incidents. Preferably, the SSG will demonstrate a worst-case scenario in a controlled environment with the permission of all involved (e.g., by showing working exploits and their business impact). In some cases, presentation to the Board can help garner resources for a new or ongoing SSI efforts. Bringing in an outside guru is often helpful when seeking to bolster executive attention. Tying education to specific development areas, such as DevOps groups using cloud-native technologies, can likewise help convince leadership to accept SSG recommendations when they might otherwise be ignored in favor of faster release dates or other priorities.

[SM1.4: 117] Implement lifecycle governance.

The software security process includes conditions for release (such as gates, checkpoints, guardrails, milestones, etc.) at one or more points in a software lifecycle. The first two steps toward establishing security-specific release conditions are to identify locations that are compatible with existing development practices and to then begin gathering the input necessary to make a go/no-go decision, such as risk ranking thresholds or defect data. Importantly, the conditions might not be verified at this stage. For example, the SSG can collect security testing results for each project prior to release, then provide their informed opinion on what constitutes sufficient testing or acceptable test results without trying to stop a project from moving forward. In CI/CD environments, shorter release cycles often require creative approaches to collecting the right evidence and rely heavily on automation. The idea of defining governance checks in the process first and enforcing them later is extremely helpful in moving development toward software security without major pain (see [SM2.2 Verify release conditions with measurements and track exceptions]). Socializing the conditions and then verifying them once most projects already know how to succeed is a gradual approach that can motivate good behavior without requiring it.

SM LEVEL 2

[SM2.1: 64] Publish data about software security internally.

To facilitate improvement, data are published internally about the state of software security within the organization. This information might come in the form of a dashboard with metrics for executives and software development management. Sometimes, these published data won't be shared with everyone in the firm but only with relevant executives who then drive change in the organization. In other cases, open book management and data published to all stakeholders help everyone know what's going on, the philosophy being that sunlight is the best disinfectant. If the organization's culture promotes internal competition between groups, this information can add a security dimension. Increasingly, security telemetry is used to gather measurements quickly and accurately, and might initially focus less on historical trends (e.g., bugs per release) and more on speed (e.g., time to fix) and quality (e.g., defect density). Some SSIs might publish these data primarily in engineering groups within pipeline platform dashboards, democratizing measurement for developer self-improvement.

[SM2.2: 61] Verify release conditions with measurements and track exceptions.

Security release conditions (or gates, checkpoints, guardrails, milestones, etc.) are verified for every project, so each project must either meet an established measure or obtain a waiver in order to move forward normally, and the SSG tracks exceptions. In some cases, measures are directly associated with regulations, contractual agreements, and other obligations, with exceptions tracked as required by statutory or regulatory drivers. In other cases, measures yield some manner of KPIs that are used to govern the process. Allowing any projects to automatically pass or granting waivers automatically without due consideration defeats the purpose of verifying conditions. Even seemingly innocuous software projects must successfully satisfy the prescribed security conditions in order to progress to or remain in production. Similarly, APIs, frameworks, libraries, bespoke code, microservices, container configurations, and so on are all software that must satisfy security release conditions. It's possible, and often very useful, to have verified the conditions both before and after the development process itself. In modern development environments, the measurement process for conditions will increasingly become automated (see [SM3.4 Integrate software-defined lifecycle governance]).

[SM2.3: 55] Create or grow a satellite.

Create a collection of people scattered across the organization who show an above-average level of security interest or skill—a satellite. Forming this group of advocates, sometimes referred to as champions, is a step toward scaling security by creating a social network that speeds the adoption of 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 Identify new satellite members through observation]. 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. A strong satellite is a good sign of a mature SSI. The satellite can act as a sounding Board for new SSG 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 agile environments, satellite-led efforts are being replaced by automation.

[SM2.6: 65] Require security sign-off.

The organization has an initiative-wide process for accepting security risk and documenting accountability, with a risk owner signing off on the state of all software prior to release. The sign-off policy might require the head of a business unit to sign-off on critical vulnerabilities that have not been mitigated or on SSDL steps that have been skipped, for example. The sign-off policy must apply both to outsourced projects and to projects that will be deployed in external environments, such as the cloud. 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, but there must be an ongoing verification that the criteria remain accurate and the automation is actually working.

SM LEVEL 3

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

The SSG uses centralized tracking automation to chart the progress of every piece of software and deployable artifact (e.g., container registries) in its purview, 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 these data at least among executives. Depending on the culture, this can cause interesting effects via internal competition. 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: 4] Run an external marketing program.

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

[SM3.3: 18] Identify metrics and use them to drive resourcing.

The SSG and its management choose the 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. Recall that, in agile methodologies, metrics are best collected early and often using event-driven processes with telemetry rather than point-in-time data collection. The key is to tie security results to business objectives in a clear and obvious fashion in order to justify funding. Because the concept of security is already tenuous to many business people, making an explicit tie-in can be helpful.

[SM3.4: 1] Integrate software-defined lifecycle governance.

Organizations begin replacing traditional document, presentation, and spreadsheet-based lifecycle management with software-based delivery platforms. Humans, sometimes aided by tools, are no longer the primary drivers of progression from each lifecycle phase to the next. Instead, organizations rely on automation to drive the management and delivery process with ALM/ADLM software, such as Spinnaker or pipeline platform software like GitHub, and humans participate asynchronously (and often optionally), like services. Automation often extends beyond the scope of CI/CD to include functional and nonfunctional aspects of delivery, including health checks, cut-over on failure, rollback to known-good software, 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 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 (e.g., output from sensors embedded in development and production). Lifecycle governance extends beyond defect discovery, and often includes incorporation of intelligence feeds and third-party security research, vulnerability disclosure and patching processes, as well as other activities.

GOVERNANCE: COMPLIANCE & POLICY (CP)

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

CP LEVEL 1

[CP1.1: 94] Unify regulatory pressures.

If the business or its customers are subject to regulatory or compliance drivers such as PCI security standards; GLBA, SOX, and HIPAA in the US; or GDPR in the EU, the SSG acts as a focal point for understanding the constraints such drivers impose on software. In some cases, the SSG creates or collaborates on a unified approach that removes redundancy and conflicts from overlapping compliance requirements. A formal approach will map applicable portions of regulations to controls applied to software to explain how the organization complies. As an alternative, existing business processes run by legal or other risk and compliance groups outside the SSG could also serve as the regulatory focal point. A unified set of software security guidance for meeting regulatory pressures ensures that compliance work is completed as efficiently as possible. Some firms move on to influencing the regulatory environment directly by becoming involved in standards groups exploring new technologies and mandates.

[CP1.2: 112] Identify PII obligations.

The SSG plays a key role in identifying and describing PII obligations stemming from regulation and customer expectations by using this information to promote best practices related to privacy. The way software handles PII might be explicitly regulated, but even if it isn't, privacy is an important topic. For example, if the organization processes credit card transactions, the SSG will help in identifying the constraints that the PCI DSS places on the handling of cardholder data and then inform all stakeholders. Note that outsourcing to hosted environments (e.g., the cloud) doesn't relax PII obligations and can even increase the difficulty of recognizing all associated obligations. Also note that firms creating software products that process PII (where those firms don't necessarily handle PII directly) might meet this need by providing privacy controls and guidance for their customers. Evolving consumer privacy expectations, the proliferation of "software is in everything," and data scraping and correlation (e.g., social media) add additional expectations for PII protection.

[CP1.3: 86] Create policy.

The SSG guides the rest of the organization by creating or contributing to software security policy that satisfies internal, regulatory, and customer-driven security requirements. This policy is what is permitted and denied at the initiative level; if it's not mandatory, it's not policy. It includes a unified approach for satisfying the (potentially lengthy) list of security drivers at the governance level. As a result, 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 the umbrella. In many cases, policy statements are translated into automation to provide governance-as-code, not just a process enforced by humans, but even policy that's been automated must be mandatory. In some cases, policy will be documented exclusively as governance-as-code, often as tool configuration, but must still be readily readable, auditable, and editable by humans. In some cases, satellite practitioners or similar roles are innovating and driving SSI policy locally in engineering. Because they might be new topics, codifying decisions about, for example, cloud-native technologies can rekindle interest in setting policy. Similarly, it might be necessary, for example, to explain what can and can't be automated into CI/CD pipelines (see [SM3.4 Integrate software-defined lifecycle governance]).

CP LEVEL 2

[CP2.1: 55] Build 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. A PII inventory might be approached in two ways: starting with each individual application by noting its PII use or starting with particular types of PII and noting the applications that touch them. System architectures have evolved such that PII will flow into cloud-based service and endpoint device ecosystems, and come to rest there (e.g., content delivery networks, social networks, mobile devices, IoT devices), making it tricky to keep an accurate PII inventory. In general, simply noting which applications process PII isn't enough; the type of PII and where it is stored are necessary so the inventory can be easily referenced in critical situations. This usually includes making a list of databases that would require customer notification if breached or a list to use in crisis simulations (see [CMVM3.3 Simulate software crises]).

[CP2.2: 47] Require security sign-off for compliance-related risk.

The organization has a formal compliance risk acceptance and accountability process that addresses all software development projects. In that process, the SSG acts as an advisor while the risk owner signs off on the software's state prior to release based on adherence to documented criteria. For example, the sign-off policy might require the head of the business unit to sign off on compliance issues that haven't been mitigated or on compliance-related SSDL steps that have been skipped. Sign-off is explicit and captured for future reference, with any exceptions tracked, even under the fastest of application lifecycle methodologies. Note that an application without security defects might still be noncompliant ,so a clean penetration test is 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 criteria are embedded in automation. 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 an ongoing verification that the criteria remain accurate and the automation is actually working.

[CP2.3: 61] Implement and track controls for compliance.

The organization can demonstrate compliance with applicable requirements because its SSDL is aligned with the control statements developed by the SSG in collaboration with compliance stakeholders (see [CP1.1 Unify regulatory pressures]). The SSG collaborates with stakeholders to track controls, navigate problem areas, and ensure auditors and regulators are satisfied. The SSG might be able to remain in the background because following the SSDL automatically generates the desired compliance evidence predictably and reliably. Increasingly, the DevOps approach of embedding compliance controls in automation shows up within 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: 48] Include software security SLAs in all vendor contracts.

Software vendor contracts include an SLA to ensure the vendor won't jeopardize the organization's compliance story or SSI. 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 (see [SR2.5 Create SLA boilerplate]). 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. Typical provisions set requirements for policy conformance, incident management, training, and defect management. Traditional IT security requirements and a simple agreement to allow penetration testing aren't sufficient here.

[CP2.5: 70] Ensure executive awareness of compliance and privacy obligations.

To gain executive buy-in around compliance and privacy activities, the SSG provides executives with plain-language explanations of the organization's compliance and privacy obligations, along with the potential consequences of failing to meet those obligations. For some organizations, explaining the direct cost and likely fallout from a compliance failure or data breach can be an effective way to broach the subject. For others, having an outside expert address the Board works because some executives value an outside perspective more than an internal one. A sure sign of proper executive buy-in is an acknowledgment of the need and adequate allocation of resources to meet those obligations. While useful for bootstrapping efforts, be aware that the sense of urgency typically following a breach will not last.

CP LEVEL 3

[CP3.1: 26] Create a regulator compliance story.

The SSG has information regulators want, so a combination of written policy, controls documentation, and artifacts gathered through the SSDL gives the SSG the ability to demonstrate the organization's compliance story without a fire drill for every audit. Often, regulators, auditors, and senior management 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 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. While they are often the biggest, governments aren't the only regulators of behavior.

[CP3.2: 16] Impose policy on vendors.

Vendors are required to adhere to the same policies used internally and must submit evidence that their software security practices pass muster. For a given organization, vendors might comprise 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. Evidence of their compliance could include results from SSDL activities or from tests built directly into automation or infrastructure. Vendors might attest to the fact that they perform certain SSDL processes. Policy enforcement might be through a point-in-time review (like that which assures acceptance criteria), enforced by automated checks (such as are applied to pull requests, committed artifacts like containers, or similar), or a matter of convention and protocol (e.g., services cannot connect unless particular security settings are correct, identifying certificates present, and so forth).

[CP3.3: 8] Drive feedback from software lifecycle data back to policy.

Information from the software lifecycle is routinely fed back into the policy creation and maintenance process to prevent defects from occurring in the first place and to help strengthen governance-as-code practices. With this process, blind spots can be eliminated by mapping them to trends in SSDL failures. The regular appearance of inadequate architecture analysis, recurring vulnerabilities, ignored security release conditions, or the wrong firm choice for carrying out a penetration test can expose policy weakness. As an example, lifecycle data might indicate that policies impose too much bureaucracy by introducing friction that prevents engineering from meeting the expected delivery cadence. Rapid technology evolution might also create policy gaps that must be addressed. Over time, policies become more practical and easier to carry out (see [SM1.1 Publish process and evolve as necessary]). Ultimately, policies are refined with SSDL data to enhance and improve a firm's effectiveness.

GOVERNANCE: TRAINING (T)

Training has always played a critical role in software security because software developers and architects often start with little security knowledge.

T LEVEL 1

[T1.1: 83] Conduct awareness training.

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

[T1.5: 38] Deliver role-specific advanced curriculum.

Software security training goes beyond building awareness by enabling students to incorporate security practices into their work. This training is tailored to cover the tools, technology stacks, development methodologies, and bugs that are most relevant to the students. For example, an organization could offer tracks for its engineers: one each for architects, developers, operations, site reliability engineers, and testers. Tool-specific training is also commonly needed in such a curriculum. While perhaps more concise than engineering training, role-specific training is necessary for many stakeholders within an organization, including product management, executives, and others.

[T1.7: 53] Deliver on-demand individual training.

The organization lowers the burden on students and reduces the cost of delivering training by offering on-demand training for individuals across roles. 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 the students in various roles to achieve its intended purpose. Training that isn't used won't create any change. Hot topics like containerization and security orchestration and new delivery styles such as gamification will attract more interest than boring policy discussions. For developers, it's possible to provide training directly through the IDE right when it's needed, but in some cases, building a new skill (such as cloud security or threat modeling) might be better suited for instructor-led training, which can also be provided on demand.

[T1.8: 41] Include security resources in onboarding.

The process for bringing new hires into an engineering organization requires that they complete a training module about software security. The generic new hire process usually covers topics like picking a good password and making sure that people don't follow you into the building, but this orientation period can be enhanced to cover topics such as how to create, deploy, and operate secure code, the SSDL, and internal security resources (see [SR1.2 Create a security portal]). 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.

T LEVEL 2

[T2.5: 32] Enhance satellite through training and events.

The SSG strengthens the satellite network by inviting guest speakers or holding special events about advanced topics (e.g., the latest software security techniques for DevOps or cloud-native technologies). This effort is about providing to the satellite customized training so that it can fulfill its specific responsibilities, not about inviting the satellite members to routine brown bags or signing them up for the standard computer-based training. In addition, 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. Face-to-face meetings are by far the most effective, even if they happen only once or twice a year and some participants must attend over videoconferencing. In teams with many geographically dispersed and work-from-home members, simply turning on cameras and ensuring everyone gets a chance to speak makes a substantial difference.

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

To make a strong and lasting change in behavior, training includes material specific to the company's history. 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 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 shouldn't cover platforms not used by developers (developers orchestrating containers probably won't care about old virtualization problems) or examples of problems relevant only to languages no longer in common use (e.g., Go developers probably don't need to understand how buffer overflows happen in C).

T LEVEL 3

[T3.1: 4] Reward progression through curriculum.

Knowledge is its own reward, but progression through the security curriculum brings other benefits, too, such as 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/or human resources team can make security's impact 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: 22] Provide training for vendors and outsourced workers.

Vendors and outsourced workers receive the same level of software security training given to employees. Spending time and effort helping suppliers get security right at the outset is much easier than trying to determine what went wrong later on, especially if the development team has moved on to other projects. Training individual contractors is much more natural than training entire outsource 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, 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.

[T3.3: 16] Host software security events.

The organization highlights its security culture as a differentiator by hosting security events featuring external speakers and content. 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, and the organization benefits from putting its security credentials on display (see [SM3.2 Run an external marketing program]). Events open only to small, select groups won't result in the desired culture change across the organization.

[T3.4: 19] 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 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, but it's useful only if it's fresh. Sufficient coverage of topics and changes from the previous year will likely comprise a significant amount of content.

[T3.5: 7] Establish SSG office hours.

The SSG or equivalent stakeholder offers help to anyone during an advertised meet-up period or regularly scheduled office hours. 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, but flexible office hours are also a possibility, with visits to particular product or application groups by request, perhaps prioritizing visits by key functionality being developed and its security implications. Slack and other messaging applications can capture questions 24x7 as a good way to seed office hours conversations, but instant messages alone aren't a replacement for conversation and problem-solving.

[T3.6: 1] Identify new satellite members through observation.

Recruit future satellite members (e.g., champions) by noting people who stand out during training courses, office hours, capture-the-flag exercises, hack-a-thons, and other opportunities to show skill and enthusiasm, and them encouraging them to join the satellite. Pay particular attention to practitioners contributing code, security configuration for orchestration, 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 Create or grow a satellite]). 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: ATTACK MODELS (AM)

Attack Models capture information used to think like an attacker: threat modeling, abuse case development and refinement, data classification, and technology-specific attack patterns.

AM LEVEL 1

[AM1.2: 81] Create a data classification scheme and 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 stores according to the kinds of data processed or services called, regardless of deployment model (e.g., on- or off-premise). This allows applications to be prioritized by their data classification. 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 Build PII inventory]) and then derive classifications for applications according to the repositories they use. Other approaches to the problem include data classification according to protection of intellectual property, impact of disclosure, exposure to attack, relevance to GDPR, and geographic boundaries.

[AM1.3: 38] Identify potential attackers.

The SSG identifies potential attackers in order to understand their motivations and abilities. The outcome of this exercise could be a set of attacker profiles that includes outlines for categories of attackers and more detailed descriptions for noteworthy individuals. 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 account for the organization's evolving software supply chain and attack surface.

[AM1.5: 57] Gather and use attack intelligence.

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

AM LEVEL 2

[AM2.1: 12] Build attack patterns and abuse cases tied to potential attackers.

The SSG prepares the organization for SSDL activities by working with stakeholders to build attack patterns and abuse cases tied to potential attackers (see [AM1.3 Identify potential attackers]). However, these resources don't have to be built from scratch for every application in order to be useful; rather, standard sets might exist for applications with similar profiles, and the SSG can add to the pile based on its own attack stories. 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. If a firm tracks the fraud and monetary costs associated with particular attacks, this information can in turn be used to prioritize the process of building attack patterns and abuse cases. Evolving software architectures (e.g., zero trust, serverless) might require organizations to evolve their attack pattern and abuse case creation approach and content.

[AM2.2: 10] Create technology-specific attack patterns.

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

[AM2.5: 16] Build and maintain a top N possible attacks list.

The SSG periodically digests the ever-growing list of attack types and focuses the organization on prevention efforts for a prioritized short list—the top N—and uses it 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 perception of potential business loss while others might prioritize according to successful attacks against their software. The top N list doesn't need to be updated with great frequency, and attacks can be coarsely sorted. For example, the SSG might brainstorm twice a year to create lists of attacks the organization should be prepared to counter "now," "soon," and "someday."

[AM2.6: 10] 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 to help counter a generic approach that might be overly focused on other organizations' top 10 lists or outdated platform attacks (see [T2.8 Create and use material specific to company history]). Hiding or overly sanitizing information about attacks from people building new systems fails to garner any positive benefits from a negative happenstance.

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

The organization has an internal, interactive forum where the SSG, the satellite, incident response, and others discuss attacks and attack methods. The discussion serves to communicate the attacker perspective to everyone. The SSG can also maintain an internal mailing list that encourages subscribers to discuss the latest information on publicly known incidents. Dissection of attacks and exploits that are relevant to a firm are particularly helpful when they spur discussion of development, infrastructure, and other mitigations. Simply republishing items from public mailing lists doesn't achieve the same benefits as active discussion, nor does a closed discussion hidden from those actually creating code. Everyone should feel free to ask questions and learn about vulnerabilities and exploits (see [SR1.2 Create a security portal]).

AM LEVEL 3

[AM3.1: 3] Have a research group that develops new attack methods.

A research group works to identify and defang new classes of attacks before attackers even know that they exist. Because the security implications of new technologies might not have been fully explored in the wild, doing it in-house is sometimes the best way forward. This isn't a penetration testing team finding new instances of known types of weaknesses—it's a research group that innovates new types of attacks. Some firms provide researchers time to follow through on their discoveries using bug bounty programs or other means of coordinated disclosure. Others allow researchers to publish their findings at conferences like DEF CON to benefit everyone.

[AM3.2: 4] 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 or a disclosing third party could require a new tool, so the SSG could package the tool and distribute it to testers. The idea here is to push attack capability past what typical commercial tools and offerings encompass, and then make that knowledge and technology easy for others to use. Tailoring these new tools to a firm's particular technology stacks and potential attackers increases the overall benefit. 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 Implement event-driven security testing in automation]).

[AM3.3: 4] Monitor automated asset creation.

The SSG guides the implementation of technology controls that provide a continuously updated view of the various network, machine, software, and related infrastructure assets being instantiated by engineering teams as part of their ALM processes. To help ensure proper coverage, the SSG works with engineering teams to understand orchestration, cloud configuration, and other self-service means of software delivery used to quickly stand-up servers, databases, networks, and entire clouds for software deployments. Monitoring the changes in application design (e.g., moving a monolithic application to microservices) is also part of this effort. This monitoring requires a specialized effort—normal system, network, and application logging and analysis won't suffice. Success might require a multi-pronged approach, including consuming orchestration and virtualization metadata, querying cloud service provider APIs, and outside-in web crawling and scraping. As processes improve, the data will be helpful for threat modeling efforts (see [AA1.1 Perform security feature review]).

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 middleware frameworks for those controls, and creating and publishing proactive security guidance.

SFD LEVEL 1

[SFD1.1: 102] Integrate and deliver security features.

Rather than having each project team implement its own security features (e.g., authentication, role management, key management, logging, cryptography, protocols), the SSG provides proactive guidance by acting as or facilitating a clearinghouse of security features for engineering groups to use. These features might be discovered during SSDL activities, created by the SSG or specialized development teams, or defined in configuration templates (e.g., cloud blueprints) and delivered via mechanisms such as containers, microservices, and APIs. Generic security features often have to be tailored for specific platforms. For example, each mobile and cloud platform will likely need their own means by which users are authenticated and authorized, secrets are managed, and user actions are centrally logged and monitored. Project teams benefit from implementations that come preapproved by the SSG, and the SSG benefits by not having to repeatedly track down the kinds of subtle errors that often creep into security features.

[SFD1.2: 76] Engage the SSG with architecture teams.

Security is a regular topic in the organization's software architecture discussions, with the architecture team taking responsibility for security in the same way that it takes responsibility for performance, availability, scalability, and resiliency. One way to keep security from falling out of these discussions is to have an SSG member participate in architecture discussions. Increasingly, architecture discussions include developers and site reliability engineers governing all types of software components, such as open source, APIs, containers, and cloud services. In other cases, enterprise architecture teams can help the SSG create secure designs that integrate properly into corporate design standards. Proactive engagement by the SSG is key to success here. Moving a well-known system to the cloud means reengaging the SSG, for example. It's never safe for one team to assume another team has addressed security requirements.

SFD LEVEL 2

[SFD2.1: 32] Leverage secure-by-design components and services.

The SSG takes a proactive role in software design by building or providing pointers to secure-by-design software components and services. In addition to teaching by example, these resilient building blocks aid important efforts such as architecture analysis and code review by making it easier to spot errors and avoid mistakes. These components and services, whether created internally or available from service providers, often have features (e.g., application identity, RBAC) that enable uniform security orchestration across, for example, multi-environment deployments. Similarly, the SSG might further leverage this information by tailoring code review rules specifically for the components it offers (see [CR2.6 Use automated tools with tailored rules]). When integrating software components, including open source and cloud services, the SSG must carefully vet the software for security before publication.

[SFD2.2: 51] Create capability to solve difficult design problems.

The SSG contributes 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 (e.g., feature velocity). Involving the SSG in the design of a new protocol, microservice, or architecture decision (e.g., containerization) enables timely analysis of the security implications of existing defenses and identifies elements that should be refactored, duplicated, or avoided. Likewise, having a security architect understand the security implications of moving a seemingly well-understood application to the cloud saves a lot of headaches later. Designing for security up front is more efficient than analyzing an existing design for security and refactoring when flaws are uncovered, so the SSG should be involved early in the new project process. The SSG could also get involved in what could have historically been purely engineering discussions, as even rudimentary (e.g., “Hello, world!”) use of cloud-native technologies requires configurations and other capabilities that have direct implications on security posture. Note that some design problems will require specific expertise outside of the SSG: even the best expert can’t scale to cover the needs of an entire software portfolio.

SFD LEVEL 3

[SFD3.1: 14] Form a review board or central committee to approve and maintain secure design patterns.

A review board or central committee formalizes the process of reaching consensus on design needs and security tradeoffs. Unlike a typical architecture committee focused on functions, this group focuses on providing security guidance and also periodically reviews already published design standards (especially around authentication, authorization, and cryptography) to ensure that design decisions don’t become stale or out of date. A review board can help control the chaos associated with adoption of new technologies when development groups might otherwise make decisions on their own without engaging the SSG. Review board security guidance also serves to inform outsourced software providers about security expectations (see [CP3.2 Impose policy on vendors]).

[SFD3.2: 14] Require use of approved security features and frameworks.

Implementers take their security features and frameworks from an approved list or repository. 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. Essentially, the more a project uses proven components, the easier testing, code review, and architecture analysis become (see [AA1.1 Perform security feature review]). 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 facilitates delivering services as software features (e.g., identity-aware proxies). Containerization makes it especially easy to package and reuse approved features and frameworks (see [SE2.5 Use application containers]).

[SFD3.3: 4] Find and publish secure design patterns from the organization.

The SSG fosters 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 could promote positive elements identified during threat modeling or architecture analysis so that good ideas are spread. This process is formalized: an ad hoc, accidental noticing isn’t sufficient. In some cases, a central architecture or technology team can facilitate and enhance this activity. Common design patterns accelerate development, so it’s important to use secure design patterns not just for applications but for all software assets (microservices, APIs, containers, infrastructure, and automation).

INTELLIGENCE: STANDARDS & REQUIREMENTS (SR)

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

SR LEVEL 1

[SR1.1: 93] Create security standards.

The SSG meets the organization's demand for security guidance by creating standards that explain the required way to adhere to policy and carry out specific security-centric operations. A standard might describe how to perform identity-based application authentication or how to determine the authenticity of a software update, 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 (e.g., infrastructure-as-code), and service mesh configuration. Standards can be deployed in a variety of ways to keep them actionable and relevant. They can be automated into development environments (e.g., worked into an IDE or toolchain), or they can be 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: 90] 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 that people refer to for the latest and greatest on security standards and requirements, as well as for other resources provided by the SSG (e.g., training). An interactive wiki is better than a static portal with guideline documents that rarely change. Organizations can supplement these materials with mailing lists, chat channels, and face-to-face meetings. Development teams are increasingly putting software security knowledge directly into toolchains and automation that is outside the organization (e.g., GitHub), but that does not remove the need for SSG-led knowledge management.

[SR1.3: 94] Translate compliance constraints to requirements.

Compliance constraints are translated into software requirements for individual projects and are 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 routinely builds software that processes credit card transactions, PCI DSS compliance plays a role in the SSDL during the 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 or deployment artifact specifications.

SR LEVEL 2

[SR2.2: 64] Create a standards review board.

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

[SR2.4: 60] Identify open source.

Open source components included in the software portfolio and integrated at runtime are identified and reviewed to understand their dependencies. Organizations use a variety of tools and metadata provided by delivery pipelines to discover old versions of components with known vulnerabilities or that their software relies on multiple versions of the same component. Automated tools for finding open source, whether whole components or large chunks of borrowed code, are one way to approach this activity. Some software development pipeline platforms, container registries, and middleware platforms such as RedHat's OpenShift, have begun to provide this visibility as metadata resulting from behind-the-scenes artifact scanning. An informal annual review or a process that relies solely on developers asking for permission does not generate satisfactory results. Some organizations combine composition analysis results from multiple phases of the software lifecycle in order to get a more complete and accurate view of the open source being included.

[SR2.5: 44] 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. The legal department understands that the boilerplate also helps prevent compliance and privacy problems. Under the agreement, vendors and outsource providers must meet company-mandated software security standards (see [CP2.4 Include software security SLAs in all vendor contracts]). Boilerplate language might call for objective third-party insight into software security efforts, such as BSIMMsc measurements or BSIMM scores.

SR LEVEL 3

[SR3.1: 30] 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. The use of open source could be restricted to predefined projects or to a short list of open source versions that have been through an approved security screening process, have had unacceptable vulnerabilities remediated, and are made available only through specific internal repositories and containers. For some use cases, policy might preclude any use of open source. The legal department often spearheads additional open source controls due to the "viral" license problem associated with GPL code. SSGs that partner with and educate the legal department on open source security risks 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: 9] Communicate standards to vendors.

The SSG works with vendors to educate them and promote the organization's security standards. A healthy relationship with a vendor isn't guaranteed through contract language alone, so the SSG should engage with vendors, discuss vendor security practices, and explain in concrete terms (rather than legalese) what the organization expects of its vendors. 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 Integrate and deliver security features]). 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: 9] Use secure coding standards.

Secure coding standards help the organization's developers avoid the most obvious bugs and provide ground rules for code review. These standards are necessarily specific to a programming language or platform, and they can address the use of popular frameworks, APIs, libraries, and infrastructure automation. Platforms might include mobile or IoT runtimes, cloud service provider APIs, orchestration runtimes, and SaaS platforms (e.g., Salesforce, SAP). If the organization already has coding standards for other purposes, its secure coding standards should build upon them. A clear set of secure coding standards is a good way to guide both manual and automated code review, as well as to provide relevant examples for security training. Some groups might choose to integrate their secure coding standards directly into automation. While enforcement isn't the point at this stage (see [CR3.5 Enforce secure coding standards]), violation of the standards is a good teachable moment for all stakeholders. Socializing the benefits of following the standards is also a good first step to gaining widespread acceptance.

[SR3.4: 25] Create standards for technology stacks.

The organization standardizes on specific technology stacks. This means 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, AMIs, etc.) for each technology stack, further reducing the amount of work required to use the stack safely. In cloud environments, hardened configurations likely include up-to-date security patches, security configuration, and security services, such as logging and monitoring. In traditional on-premise IT deployments, a stack might include an operating system, a database, an application server, and a runtime environment (e.g., a LAMP stack). Where the technology will be reused, such as containers, microservices, or orchestration code, the security frontier is a good place to find traction; standards for secure use of these reusable technologies means that getting security right in one place positively impacts the security posture of all downstream dependencies (see [SE2.5 Use application containers]).

SSDL 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 STRIDE or Architecture Risk Analysis), and building an assessment and remediation plan for the organization.

AA LEVEL 1

[AA1.1: 114] Perform security feature review.

When getting started in architecture analysis, organizations center the process on a review of security features. Security-aware reviewers identify the security features in an application and its deployment configuration (authentication, access control, use of cryptography, etc.), and then inspect the design and runtime parameters for problems that would cause these features to fail at their purpose or otherwise prove insufficient. For example, this kind of review 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 put PII in local storage. In some cases, use of the firm's secure-by-design components can streamline this process (see [SFD2.1 Leverage secure-by-design components and services]). 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, third-party SaaS, and so on. Some of these environments might provide robust security feature sets, whereas others might have key capability gaps that require careful consideration, so organizations are not just considering the applicability and correct use of security features in one tier of the application but across all tiers that constitute the architecture and operational environment.

[AA1.2: 41] Perform design review for high-risk applications.

The organization learns the benefits of AA by seeing real results for a few high-risk, high-profile applications. Reviewers must have some experience performing detailed design reviews and breaking the architecture under consideration, especially for new platforms or environments. In all cases, a design review should produce a set of architecture flaws and a plan to mitigate them. If the SSG isn't yet equipped to perform an in-depth AA, it can use consultants to do this work, but it should participate actively. Ad hoc review paradigms that rely heavily on expertise can be used here, but they don't tend to scale in the long run. A review focused only on whether a software project has performed the right process steps won't generate useful results about architecture flaws. Note that a sufficiently robust design review process can't be executed at CI/CD speed.

[AA1.3: 32] Have SSG lead design review efforts.

The SSG takes a lead role in AA by performing a design review to uncover flaws. Breaking down an architecture is enough of an art that the SSG must be proficient at it before it can turn the job over to architects, and proficiency requires practice. 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 the detailed review with a minimum of interaction with the project team. Over time, the responsibility for leading review efforts should shift toward software security architects. Approaches to AA evolve over time, so it's wise to not expect to set a process and use it forever.

[AA1.4: 67] Use a risk methodology to rank applications.

To facilitate security feature and design review processes, the SSG or other assigned groups use a defined risk methodology, which might be implemented via questionnaire or similar method—whether manual or automated—to collect information about each application in order to assign a risk classification and associated prioritization. Information needed for an assignment 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, where the process should be short enough to take only a few minutes. Some teams might use automation to gather the necessary data. The SSG can use the answers to categorize the application as, for example, high, medium, or low risk. Because a risk questionnaire can be easy to game, it’s important to put into place some spot-checking for validity and accuracy. An overreliance on self-reporting or automation can render this activity useless.

AA LEVEL 2

[AA2.1: 23] Define and use AA process.

The SSG defines and documents a process for AA and applies it in the design reviews it conducts to find flaws. This process includes a standardized approach for thinking about attacks, vulnerabilities, and various security properties. In addition to the technical impact discussions, the process includes a focus on the associated risk, such as through frequency or probability analysis, that gives stakeholders the information necessary to make decisions. 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 people can understand and use. Microsoft’s STRIDE and Synopsys’s ARA are examples of such a process, although even these two methodologies for AA have evolved greatly over time. Individual ad hoc approaches to AA don’t count as a defined process.

[AA2.2: 24] Standardize architectural descriptions.

Defined AA processes use an agreed-upon format to describe architecture, including a means for representing data flow. Combining a documented process along with standardized architecture descriptions will make AA tractable for people who aren’t security experts. High-level network diagrams, data flow, and authorization flows are always useful, but the description should 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 whiteboard squiggles are especially useful, too.

AA LEVEL 3

[AA3.1: 11] Have engineering teams lead AA process.

Engineering teams lead the AA process most of the time. This effort requires a well-understood and well-documented process (see [AA2.1 Define and use AA process]), although the SSG still might contribute to AA in an advisory capacity or under special circumstances. Even with a good process, consistency is difficult to attain because breaking architecture requires experience, so provide architects with SSG or outside expertise on novel issues. In any given organization, the identified engineering team might normally have responsibilities such as development, DevOps, cloud security, operations security, security architecture, or a variety of similar roles.

[AA3.2: 1] Drive analysis results into standard architecture patterns.

Failures identified during AA are fed back to engineering teams so that similar mistakes can be prevented in the future through improved design patterns (see [SFD3.1 Form a review board or central committee to approve and maintain secure design patterns]). Cloud service 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 who 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) in making their own. Note that security design patterns can interact in surprising ways that break security, so the AA process should be applied even when vetted design patterns are in standard use.

[AA3.3: 6] Make the SSG available as an AA resource or mentor.

To build an AA capability outside of the SSG, the SSG advertises itself as a resource or mentor for teams that ask for help in using the AA process (see [AA2.1 Define and use AA process]) to conduct their own design reviews. The SSG might answer AA questions during office hours and, in some cases, might assign someone to sit with the architect 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.

SSDL TOUCHPOINTS: CODE REVIEW (CR)

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

CR LEVEL 1

[CR1.2: 79] Perform opportunistic code review.

The SSG ensures code review for high-risk applications is performed in an opportunistic fashion, such as by following up a design review with a code review looking for security issues in not only source code and dependencies but also deployment artifact configuration (e.g., containers) and automation metadata (e.g., infrastructure-as-code). This informal targeting often evolves into a systematic approach. Code review could involve the use of specific tools and services, or it might be manual, 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: 100] Use automated tools along with manual review.

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

[CR1.5: 49] Make code review mandatory for all projects.

Code review is mandatory for all projects under the SSG's purview, 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, 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.

[CR1.6: 41] Use centralized reporting to close the knowledge loop and drive training.

The bugs found during code review are tracked in a centralized repository that makes it possible to do both summary and trend reporting for the organization. The code review information can be incorporated into a CISO-level dashboard that might include feeds from other parts of the security organization (e.g., penetration tests, security testing, black-box testing, and white-box testing). Given the historical code review data, the SSG can also use the reports to demonstrate progress and drive the training curriculum (see [SM2.5 Identify metrics and use them to drive budgets]). Individual bugs make excellent training examples. Some organizations have moved toward analyzing this data and using the results to drive automation.

[CR1.7: 50] Assign tool mentors.

Mentors are available to show developers how to get the most out of code review tools. If the SSG has the most skill with the tools, it could use office hours or other outreach to help developers establish the right configuration or get started on interpreting results. Alternatively, someone from the SSG 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 tools isn't the same as mentoring. Increasingly, mentorship extends to tools associated with deployment artifacts (e.g., container security) and infrastructure (e.g., cloud configuration). In many organizations, satellite members (e.g., champions) take on the tool mentorship role.

CR LEVEL 2

[CR2.6: 23] Use automated tools with tailored rules.

Customize static analysis to improve efficiency and reduce false positives. Adding custom rules can help uncover security defects specific to the organization's coding standards or the framework-based or cloud-provided middleware it uses. The same group that provides tool mentoring will likely spearhead the customization. Tailored rules can be 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. To reduce the workload for everyone, many organizations also create rules to remove repeated false positives and turn off checks that aren't relevant.

[CR2.7: 24] Use a top N bugs list (real data preferred).

The SSG maintains a living list of the most important kinds of bugs that it wants to eliminate from the organization's code and uses it to drive change. Many organizations start with a generic list pulled from public sources, but lists such as the OWASP Top 10 rarely reflect an organization's bug priorities. The list's value comes from being specific to the organization, built from real data gathered from code review, testing, software composition analysis, and actual incidents, and prioritized for prevention efforts. Simply sorting the day's bug data by number of occurrences won't produce a satisfactory list because these data change 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 only include known problems. Of course, just building the list won't accomplish anything; everyone has to use it to fix bugs.

[CR3.2: 7] Build a capability to combine assessment results.

Combine assessment results so that multiple analysis techniques feed into one reporting and remediation process. In addition to code review, analysis techniques might include dynamic analysis, software composition analysis, container scanning, cloud services monitoring, and so on. The SSG might write scripts 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 use conflicting terminology. In some cases, using a standardized taxonomy (e.g., a CWE-like approach) can help with normalization. Combining multiple sources helps drive better-informed risk mitigation decisions.

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

When a new kind of bug is discovered in the firm's software, the SSG ensures rules are created to find it and helps use these rules to identify all occurrences of the new bug throughout the codebases and runtime environments. It becomes possible to eradicate the bug type entirely without waiting for every project to reach the code review portion of its lifecycle. 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 using cloud configuration tools to provide guardrails can often help in eradication efforts.

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

Automated code review is used to identify dangerous code written by malicious in-house developers or outsource providers. Examples of malicious code that could be targeted 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 code patterns in the organization's codebase will quickly become a necessity. Manual code 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 have a tendency to stay in deployed code and should be treated as malicious code until proven otherwise.

[CR3.5: 1] Enforce secure coding standards.

The enforced portions of an organization's secure coding standards often start out as a simple list of banned functions, with a violation of these standards being sufficient grounds for rejecting a piece of code. Other useful coding standard topics might include proper use of cloud APIs, use of approved cryptography, memory sanitization, and many others. 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 using automation. Standards can be positive ("do it this way") or negative ("do not use this API"), but they must be enforced to be useful.

SSDL TOUCHPOINTS: SECURITY TESTING (ST)

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

ST LEVEL 1

[ST1.1: 104] Ensure QA performs edge/boundary value condition testing.

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 understands the value of pushing past standard functional testing that uses expected input, it begins to move slowly toward thinking like an adversary. A discussion of boundary value testing can lead naturally to the notion of an attacker probing the edges on purpose (for example, determining what happens when someone enters the wrong password over and over).

[ST1.3: 89] Drive tests with security requirements and security features.

QA targets declarative security mechanisms with tests derived from requirements and security features. A test could try to access administrative functionality as an unprivileged user, for example, or verify that a user account becomes locked after some number of failed authentication attempts. For the most part, security features can be tested in a fashion similar to other software features—security mechanisms such as account lockout, transaction limitations, entitlements, and so on are tested with both expected and unexpected input as derived from 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.

ST LEVEL 2

[ST2.1: 42] Integrate black-box security tools into the QA process.

The organization uses one or more black-box security testing tools as part of the QA process. Such tools are valuable because they encapsulate an attacker's perspective, albeit generically. Traditional dynamic analysis scanners are relevant for web applications, while similar tools exist for cloud environments, containers, mobile applications, embedded systems, and so on. In some situations, other groups might collaborate with the SSG to apply the tools. For example, a testing team could run the tool but come to the SSG for help interpreting the results. When testing is integrated into agile development approaches, black-box tools might be hooked into internal toolchains, might be provided by cloud-based toolchains, or be used directly by engineering. Regardless of who runs the black-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: 20] Share security results with QA.

The SSG or others with security testing data routinely share results from security reviews with those responsible for QA. Using security testing results as the basis for a conversation about common attack patterns or the underlying causes of security defects allows QA to generalize that information into new test approaches. Organizations that have chosen to leverage software pipeline platforms such as GitHub, or CI/CD platforms such as the Atlassian stack, can benefit from QA receiving various testing results automatically, which should then facilitate timely stakeholder conversations. In some cases, these platforms can be used to integrate QA into automated remediation workflow and reporting by generating change request tickets for developers, lightening the QA workload. Over time, QA learns the security mindset, and the organization benefits from an improved ability to create security tests tailored to the organization's code.

[ST2.5: 16] Include security tests in QA automation.

Security tests are included in an automation framework and run alongside other QA tests. While many groups trigger these tests manually, in a modern toolchain, these tests are likely part of the pipeline and triggered through automation. Security tests might be derived from abuse cases identified earlier in the lifecycle (see [AM2.1 Build attack patterns and abuse cases tied to potential attackers]), from creative tweaks of functional tests, developer tests, and security feature tests, or even from guidance provided by penetration testers on how to reproduce an issue.

[ST2.6: 13] Perform fuzz testing customized to application APIs.

QA efforts include running a customized fuzzing framework against APIs critical to the organization. They could begin from scratch or use an existing fuzzing toolkit, but the necessary customization often goes beyond creating custom protocol descriptions or file format templates to giving the fuzzing framework a built-in understanding of the application interfaces it calls into. Test harnesses developed explicitly for particular applications make good places to integrate fuzz testing.

ST LEVEL 3

[ST3.3: 5] Drive tests with risk analysis results.

Testers use architecture analysis results (see [AA 2.1 Define and use AA process]) to direct their work. If the AA determines that "the security of the system hinges on the transactions being atomic and not being interrupted partway through," for example, then torn transactions will become a primary target in adversarial testing. Adversarial tests like these can be developed according to risk profile, with high-risk flaws at the top of the list. Security testing results shared with QA (see [ST 2.4 Share security results with QA]) 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: 2] Leverage coverage analysis.

Testers measure the code coverage of their security tests (see [ST2.5 Include security tests in QA automation]) to identify code that isn't being exercised and then adjust automation (see [ST3.6 Implement event-driven security testing in automation]) to incrementally improve coverage. In turn, code coverage analysis drives increased security testing depth. Standard-issue black-box testing tools achieve exceptionally low coverage, leaving a majority of the software under test unexplored, which isn't a testing best practice. Coverage analysis is easier when using standard measurements such as function coverage, line coverage, or multiple condition coverage.

[ST3.5: 2] Begin to build and apply adversarial security tests (abuse cases).

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

[ST3.6: 0] Implement event-driven security testing in automation.

The SSG guides implementation of automation for continuous, event-driven application security testing. Event-driven testing implemented in ALM automation typically moves the testing closer to the conditions driving the testing requirement (whether shift left toward design or shift right toward operations), repeats the testing as often as the event is triggered as software moves through ALM, and helps ensure the right testing is executed for a given set of conditions. This might be instead of or in addition to software arriving at a gate or checkpoint and triggering a point-in-time security test. Success with this approach depends on broad use of sensors (e.g., agents, bots) that monitor engineering processes, execute contextual rules, and provide telemetry to automation that initiates the specified testing whenever event conditions are met. More mature configurations proceed to including risk-driven conditions.

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 the final configuration and provides direct feeds to defect management and mitigation.

PT LEVEL 1

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

Use external penetration testers to demonstrate that the organization's code needs help. Breaking a high-profile application to provide unmistakable evidence that the organization isn't somehow immune to the problem often gets the right attention. 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. External penetration testers that bring a new set of experiences and skills to the problem are the most useful.

[PT1.2: 100] Feed results to the defect management and mitigation system.

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. Testing often targets container and infrastructure configuration in addition to applications, and results are commonly provided in machine-readable formats to enable automated tracking. Properly done, this exercise demonstrates the organization's ability to improve the state of security, and many firms are emphasizing the critical 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., Slack, JIRA) to communicate change requests, but those requests are still tracked explicitly as part of a vulnerability management process.

[PT1.3: 89] Use penetration testing tools internally.

The organization creates an internal penetration testing capability that uses tools. This capability (e.g., group, team) can be part of the SSG or part of a specialized team elsewhere in the organization, with the tools complementing manual efforts to improve the efficiency and repeatability of the testing process. Tools used 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.

PT LEVEL 2

[PT2.2: 30] Penetration testers use all available information.

Penetration testers, whether internal or external, use source code, design documents, architecture analysis results, misuse and abuse cases, code review results, and cloud environment and other deployment configuration to do deeper analysis and find more interesting problems. To effectively do their job, penetration testers often need everything created throughout the SSDL, so an SSDL that creates no useful artifacts about the code will make this effort harder. Having access to the artifacts is not the same as using them.

11 New activity for BSIMM11

[PT2.3: 29] Schedule periodic penetration tests for application coverage.

The SSG collaborates in periodically testing all applications in its purview according to an established schedule, which could be tied to a calendar or a release cycle. High-profile applications should get a penetration test at least once a year, even if new releases haven't yet moved into production. 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 containers and components in the cloud. One important aspect of periodic testing is to make sure that the problems identified are actually fixed and don't creep back into the build. New automation created for CI/CD deserves penetration testing as well.

PT LEVEL 3

[PT3.1: 21] Use external penetration testers to perform deep-dive analysis.

The organization uses external penetration testers to do deep-dive analysis for critical projects and to introduce fresh thinking into the SSG. These testers should be 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. Skilled penetration testers will always break a system, but the question is whether they demonstrate new kinds of thinking about attacks that can be useful when 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: 10] Customize penetration testing tools.

The SSG collaborates in either creating penetration testing tools or adapting publicly available ones to more efficiently and comprehensively attack the organization's software. Tools will improve the efficiency of the penetration testing process without sacrificing the depth of problems that the SSG can identify. Automation can be particularly valuable in organizations using agile methodologies because it helps teams go faster. Tools that can be tailored are always preferable to generic tools. Success here is often dependent upon both the depth of tests and their scope.

DEPLOYMENT: SOFTWARE ENVIRONMENT (SE)

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

SE LEVEL 1

[SE1.1: 73] Use application input monitoring.

The organization monitors the input to the software that it runs in order to spot attacks. For web code, a WAF can do this job, while other kinds of software likely require other approaches, including runtime instrumentation. The SSG might be responsible for the care and feeding of the monitoring system, but incident response isn't part of this activity. For web applications, WAFs that write log files can be useful if someone periodically reviews the logs and takes action. Other 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: 121] Ensure host and network security basics are in place.

The organization provides a solid foundation for its software by ensuring that host and network security basics are in place across its data centers and networks and ensuring it remains in place during new releases. Evolving network perimeters, increased connectivity and data sharing, and increasing interdependence on vendors (e.g., content delivery, load balancing, and content inspection services) add a degree of difficulty even to getting the basics right. Doing software security before getting host and network security in place is like putting on shoes before putting on socks.

SE LEVEL 2

[SE2.2: 39] Define secure deployment parameters and configurations.

The SSDL requires creating an installation guide or a clearly described configuration for deployable software artifacts and the infrastructure-as-code necessary to deploy them, helping teams install and configure software securely. When special steps are required to ensure a deployment is secure, these steps can be outlined in a configuration guide or explicitly described in deployment automation, including information on COTS, vendor, and cloud services components. In some cases, installation guides are not used internally but are distributed to customers who buy the software. All deployment automation should be understandable by humans, not just by machines. Increasingly, secure deployment parameters and configuration are codified into infrastructure scripting (e.g., Terraform, Helm, Ansible, and Chef).

[SE2.4: 33] Protect code integrity.

The organization can attest to the provenance, integrity, and authorization of important code before allowing it to execute. 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 material. In some cases, organizations allow only code from their own registries to execute in certain environments. With many DevOps practices greatly increasing the number of people who can touch the code, organizations should also use permissions and peer review to govern code commits within source code management to help protect integrity.

[SE2.5: 31] Use application containers.

The organization uses application containers to support its software security goals, which likely include ease of deployment, a tighter coupling of applications with their dependencies, immutability, integrity (see [SE2.4 Protect code integrity]), and some isolation benefits without the overhead of deploying a full operating system on a virtual machine. Containers provide a convenient place for security controls to be applied and updated consistently. While containers can be useful in development and test environments, production use provides the needed security benefits.

[SE2.6: 36] Ensure cloud security basics.

Organizations should already be ensuring that their host and network security basics are in place, but they must also ensure that basic requirements are met in cloud environments. Of course, cloud-based virtual 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 implement cloud-based security features and controls (some of which can be built in, for example, cloud provider administration consoles) that are 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.

SE LEVEL 3

[SE3.2: 13] 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). This is particularly important for widely distributed code such as mobile applications and JavaScript distributed to browsers. 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. On some platforms, employing Data Execution Prevention (DEP), Safe Structured Handling (SafeSEH), and Address Space Layout Randomization (ASLR) can be a good start at making exploit development more difficult.

[SE3.3: 7] Use application behavior monitoring and diagnostics.

The organization monitors production software to look for misbehavior or signs of attack. This activity goes beyond host and network monitoring to look for software-specific problems, such as indications of malicious behavior, fraud, and related issues. Intrusion detection and anomaly detection systems at the application level 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. In any case, the signs that an application isn't behaving as expected will be specific to the software and its environment, so one-size-fits-all solutions probably won't generate satisfactory results. In some types of environments (e.g., PaaS), some of this data and the associated predictive analytics might come from a vendor.

[SE3.5: 22] 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, such as hardening against drift, secrets management, and rollbacks (see [SDF2.1 Leverage secure-by-design components and services]), to ensure 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 become 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.

[SE3.6: 12] Enhance application inventory with operations bill of materials.

A list of applications and their locations in production environments is essential information for any well-run enterprise (see [CMVM2.3 Develop an operations inventory of software delivery value streams]). In addition, a manifest detailing the components, dependencies, configurations, external services, and so on for all production software helps the organization to tighten its security posture—that is, to react with agility as attackers and attacks evolve, compliance requirements change, and the number of items to patch grows quite large. Knowing where all the components live in running software—whether they're in private data centers, in clouds, or sold as box products—allows for timely response when unfortunate events occur. Done properly, institutional use of container security solutions can make inventory efforts much simpler.

DEPLOYMENT: CONFIGURATION MANAGEMENT & VULNERABILITY MANAGEMENT (CMVM)

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

CMVM LEVEL 1

[CMVM1.1: 108] 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 regular meeting between the SSG and the incident response team can keep information flowing in both directions. Having pre-built communication channels with critical vendors (e.g., infrastructure, SaaS, PaaS) is also very important.

[CMVM1.2: 102] Identify software defects found in operations monitoring and feed them back to development.

Defects identified in production through operations monitoring are fed back to development and used to change developer behavior. The contents of production logs can be revealing (or can reveal the need for improved logging). Entering incident triage 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. Increasingly, organizations are relying on telemetry provided by agents packaged with software as part of cloud security posture monitoring, container configuration monitoring, RASP, or similar products to detect software defects or adversaries' exploration prior to exploit. In most cases, organizations must also rely on additional analysis tools (e.g., ELK, Datadog) to aggregate and correlate that avalanche of data and provide useful input to development. In the best of cases, processes in the SSDL can be improved based on operational data (see [CMVM3.2 Enhance the SSDL to prevent software bugs found in operations]).

CMVM LEVEL 2

[CMVM2.1: 94] Have emergency response.

The organization can make quick code and configuration changes when software (e.g., application, API, microservice, infrastructure) is under attack, with a rapid-response team working in conjunction with application owners, developers, operators, 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 version, deploy a new container). Often, the emergency response team is the engineering team itself. A well-defined process is a must here, but a process that has never been used might not actually work.

[CMVM2.2: 97] Track software bugs found in operations through the fix process.

Defects found in operations are entered into established defect management systems and tracked through the fix process. This capability could come in the form of a two-way bridge between bug finders and bug fixers, but make sure the loop is closed completely. Bugs can crop up in all types of deployable artifacts, deployment automation, and infrastructure configuration. Setting a security flag in the bug-tracking system can help facilitate tracking.

[CMVM2.3: 65] Develop an operations inventory of software delivery value streams.

The organization has a map of its software deployments, related containerization and orchestration automation code, and deployment automation code, along with the respective owners that contribute to business value streams. If a software asset needs to be changed, operations or DevOps teams can reliably identify both the stakeholders and all the places where the change needs to be deployed. Common components shared between multiple projects can be noted so that, when an error occurs in one application, other applications that share the same components can be fixed as well. Accurately representing 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.

CMVM LEVEL 3

[CMVM3.1: 5] Fix all occurrences of software bugs found in operations.

The organization fixes all instances of each bug found during operations—not just the small number of instances that trigger bug reports—to meet recovery, continuity, and resiliency goals. Doing this proactively requires the ability to reexamine the entire inventory of software delivery value streams when new kinds of bugs come to light (see [CR3.3 Create capability to eradicate bugs]). One way to approach this is to create a ruleset that generalizes a deployed bug into something that can be scanned for via automated code review. Use of orchestration can greatly simplify deploying the fix for all occurrences of a software bug (see [SE3.5 Use orchestration for containers and virtualized environments]).

[CMVM3.2: 11] Enhance the SSDL to prevent software bugs found in operations.

Experience from operations leads to changes in the SSDL, which can in turn be strengthened to prevent the reintroduction of bugs found during operations. To make this process systematic, incident response postmortem could include a feedback-to-SSDL step. The outcomes of the postmortem might result in changes such as tool-based policy rulesets in a CI/CD pipeline and adjustments to automated deployment configuration (see [SM3.4 Integrated software-defined lifecycle governance]). This works best when root-cause analysis pinpoints where in the software lifecycle an error could have been introduced or slipped by uncaught. 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.

[CMVM3.3: 13] Simulate software crises.

The SSG simulates high-impact software security crises to ensure software incident detection and response capabilities minimize damage. Simulations could test for the ability to identify and mitigate specific threats or, in other cases, begin with the assumption that a critical system or service is already compromised and evaluate the organization's ability to respond. Planned chaos engineering can be effective at triggering unexpected conditions during simulations. The exercises must include attacks or other software security crises at the appropriate software layer to generate useful results (e.g., at the application layer for web applications and at lower layers for IoT devices). When simulations model successful attacks, an important question to consider is the time required to clean up. Regardless, simulations must focus on security-relevant software failure and 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: 16] 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 versus 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 crowd-sourced efforts, don't constitute a bug bounty program.

[CMVM3.5: 8] Automate verification of operational infrastructure security.

The SSG works with engineering teams to facilitate a controlled self-service process that replaces some traditional IT efforts, such as application and infrastructure deployment, and includes verification of security properties (e.g., adherence to agreed-upon security hardening). Engineers now create networks, containers, and machine instances, orchestrate deployments, and perform other tasks that were once IT's sole responsibility (see [AM3.3 Monitor automated asset creation]). In facilitating this change, the organization uses machine-readable policies and configuration standards to automatically detect issues such as configuration drift and report on infrastructure that does not meet expectations (see [SE2.2 Define secure deployment parameters and configurations]). In some cases, the automation makes changes to running environments to bring them into compliance. In many cases, organizations use a single policy to manage automation in different environments, such as in multi-cloud and hybrid-cloud environments.

11 [CMVM3.6: 0] Publish risk data for deployable artifacts.

The organization collects and publishes risk telemetry about the applications, services, APIs, containers, and other software it deploys. Published information extends beyond basic software security (see [SM2.1 Publish data about software security internally]) and inventory data (see [SM3.1 Use a software asset tracking application with portfolio view]) to include information about constituency of the software (e.g., bill of materials), what group created it and how, and the risks associated with vulnerability, security controls, or other characteristics intrinsic to each artifact. This approach stimulates cross-functional coordination and helps stakeholders take informed risk management action. In some cases, much of this information is created by automated processes and associated with a registry that provides stakeholder visibility. Making a list of risks that aren't used for decision support won't achieve useful results.

APPENDIX

In this appendix, we provide some history on the BSIMM project and how it serves as a longitudinal study of software security efforts. We also describe the most recent changes to the BSIMM. To help understand how various vertical markets approach their SSIs, we also provide a scorecard showing activity observation counts per vertical market. Finally, we provide a list of the 121 BSIMM11 activities.

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 programmer can contribute to or unintentionally undermine the security of a computer system and started asking some specific questions: What kinds of bugs and flaws lead to security problems? How can we identify problems systematically?

By the middle of the following decade, there was an emerging consensus that building secure software required more than just smart individuals toiling away. Getting security right, especially across a software portfolio, means being involved in the software development process, even as the process evolves.

“Getting security right, especially across a software portfolio, means being involved in the software development process, even as the process evolves.”

Since then, practitioners have come to learn that process and developer tools alone are insufficient. Software security encompasses business, social, and organizational aspects as well.

Table 7 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 from the dataset.) BSIMM11 describes the work of 8,457 SSG and satellite people working directly in software security, impacting the security efforts of 490,167 developers.

BSIMM NUMBERS OVER TIME											
	BSIMM11	BSIMM10	BSIMM9	BSIMM8	BSIMM7	BSIMM6	BSIMM-V	BSIMM4	BSIMM3	BSIMM2	BSIMM1
FIRMS	130	122	120	109	95	78	67	51	42	30	9
MEASUREMENTS	357	339	320	256	237	202	161	95	81	49	9
2ND MEASURES	32	50	42	36	30	26	21	13	11	0	0
3RD MEASURES	12	32	20	16	15	10	4	1	0	0	0
4TH MEASURES	7	8	7	5	2	2					
SSG MEMBERS	1,801	1,596	1,600	1,268	1,111	1,084	976	978	786	635	370
SATELLITE MEMBERS	6,656	6,298	6,291	3,501	3,595	2,111	1,954	2,039	1,750	1,150	710
DEVELOPERS	490,167	468,500	415,598	290,582	272,782	287,006	272,358	218,286	185,316	141,175	67,950
APPLICATIONS	176,269	173,233	135,881	94,802	87,244	69,750	69,039	58,739	41,157	28,243	3,970
AVG. SSG AGE (YEARS)	4.32	4.53	4.13	3.88	3.94	3.98	4.28	4.13	4.32	4.49	5.32
SSG AVG. OF AVGs	2.01 / 100	1.37 / 100	1.33 / 100	1.60 / 100	1.61 / 100	1.51 / 100	1.4 / 100	1.95 / 100	1.99 / 100	1.02 / 100	1.13 / 100
FINANCIAL SERVICES	42	57	50	47	42	33	26	19	17	12	4
FINTECH	21										
ISVs	46	43	42	38	30	27	25	19	15	7	4
TECH	27	20	22	16	14	17	14	13	10	7	2
HEALTHCARE	14	16	19	17	15	10					
INTERNET OF THINGS	17	13	16	12	12	13					
CLOUD	30	20	17	16	15						
INSURANCE	14	11	10	11	10						
RETAIL	8	9	10								

Table 7. BSIMM Numbers Over Time. The chart shows how the BSIMM experiment has grown over the years.

THE BSIMM AS A LONGITUDINAL STUDY

Fifty-three of the 130 firms in BSIMM11 have been measured at least twice. On average, the time between first and second measurements for those 53 firms was 29.8 months. Although individual activities among the 12 practices come and go (as shown in Table 8), in general, remeasurement over time shows a clear trend of increased maturity. The raw score went up in 46 of the 53 firms and remained the same in three firms. Across all 53 firms, the score increased by an average of 11.2 (44.9%). Simply put, SSIs mature over time.

GOVERNANCE			INTELLIGENCE			SSDL TOUCHPOINTS			DEPLOYMENT		
ACTIVITY	BSIMM ROUND 1 (OF 53)	BSIMM ROUND 2 (OF 53)	ACTIVITY	BSIMM ROUND 1 (OF 53)	BSIMM ROUND 2 (OF 53)	ACTIVITY	BSIMM ROUND 1 (OF 53)	BSIMM ROUND 2 (OF 53)	ACTIVITY	BSIMM ROUND 1 (OF 53)	BSIMM ROUND 2 (OF 53)
[SM1.1]	25	43	[AM1.2]	35	44	[AA1.1]	49	50	[PT1.1]	46	51
[SM1.2]	22	35	[AM1.3]	13	22	[AA1.2]	12	18	[PT1.2]	30	41
[SM1.3]	29	36	[AM1.5]	21	27	[AA1.3]	10	13	[PT1.3]	31	39
[SM1.4]	46	49	[AM2.1]	4	7	[AA1.4]	24	35	[PT2.2]	12	8
[SM2.1]	18	33	[AM2.2]	4	5	[AA2.1]	5	12	[PT2.3]	13	14
[SM2.2]	16	23	[AM2.5]	4	8	[AA2.2]	3	7	[PT3.1]	4	4
[SM2.3]	19	34	[AM2.6]	6	5	[AA3.1]	4	6	[PT3.2]	1	3
[SM2.6]	19	27	[AM2.7]	4	7	[AA3.2]	1	1			
[SM3.1]	8	17	[AM3.1]	2	1	[AA3.3]	5	5			
[SM3.2]	2	2	[AM3.2]	0	0						
[SM3.3]	9	15	[AM3.3]	0	1						
[SM3.4]	0	0									
[CP1.1]	32	40	[SFD1.1]	39	45	[CR1.2]	31	35	[SE1.1]	25	33
[CP1.2]	46	48	[SFD1.2]	34	38	[CR1.4]	33	45	[SE1.2]	45	50
[CP1.3]	25	41	[SFD2.1]	8	18	[CR1.5]	14	24	[SE2.2]	18	17
[CP2.1]	16	28	[SFD2.2]	18	24	[CR1.6]	17	29	[SE2.4]	8	13
[CP2.2]	17	19	[SFD3.1]	3	9	[CR1.7]	9	23	[SE2.5]	1	6
[CP2.3]	16	23	[SFD3.2]	5	10	[CR2.6]	6	14	[SE2.6]	0	7
[CP2.4]	17	24	[SFD3.3]	2	3	[CR2.7]	9	14	[SE3.2]	4	4
[CP2.5]	22	29				[CR3.2]	1	3	[SE3.3]	5	4
[CP3.1]	9	16				[CR3.3]	0	2	[SE3.5]	0	3
[CP3.2]	10	13				[CR3.4]	0	0	[SE3.6]	0	3
[CP3.3]	1	4				[CR3.5]	0	2			
[T1.1]	34	40	[SR1.1]	32	43	[ST1.1]	39	44	[CMVM1.1]	45	47
[T1.5]	9	20	[SR1.2]	35	42	[ST1.3]	40	40	[CMVM1.2]	48	45
[T1.7]	20	29	[SR1.3]	34	43	[ST2.1]	16	19	[CMVM2.1]	43	45
[T1.8]	9	14	[SR2.2]	14	31	[ST2.4]	4	8	[CMVM2.2]	37	43
[T2.5]	7	16	[SR2.4]	10	23	[ST2.5]	2	7	[CMVM2.3]	24	35
[T2.8]	10	8	[SR2.5]	12	22	[ST2.6]	6	4	[CMVM3.1]	1	0
[T3.1]	1	3	[SR3.1]	4	10	[ST3.3]	2	2	[CMVM3.2]	2	6
[T3.2]	3	9	[SR3.2]	7	9	[ST3.4]	0	0	[CMVM3.3]	3	3
[T3.3]	0	5	[SR3.3]	11	9	[ST3.5]	2	3	[CMVM3.4]	1	7
[T3.4]	1	12	[SR3.4]	14	16	[ST3.6]	0	0	[CMVM3.5]	0	0
[T3.5]	0	6							[CMVM3.6]	0	0
[T3.6]	2	2									

Table 8. BSIMM11 Reassessments Scorecard Round 1 vs. Round 2. The chart shows how 53 SSIs changed between assessments.

As with earlier spider diagrams, Figure 10 shows the average high-water marks per practice for the 53 firms in their first and second assessments. We see growth in nearly every practice, with the Penetration Testing practice being the possible exception.

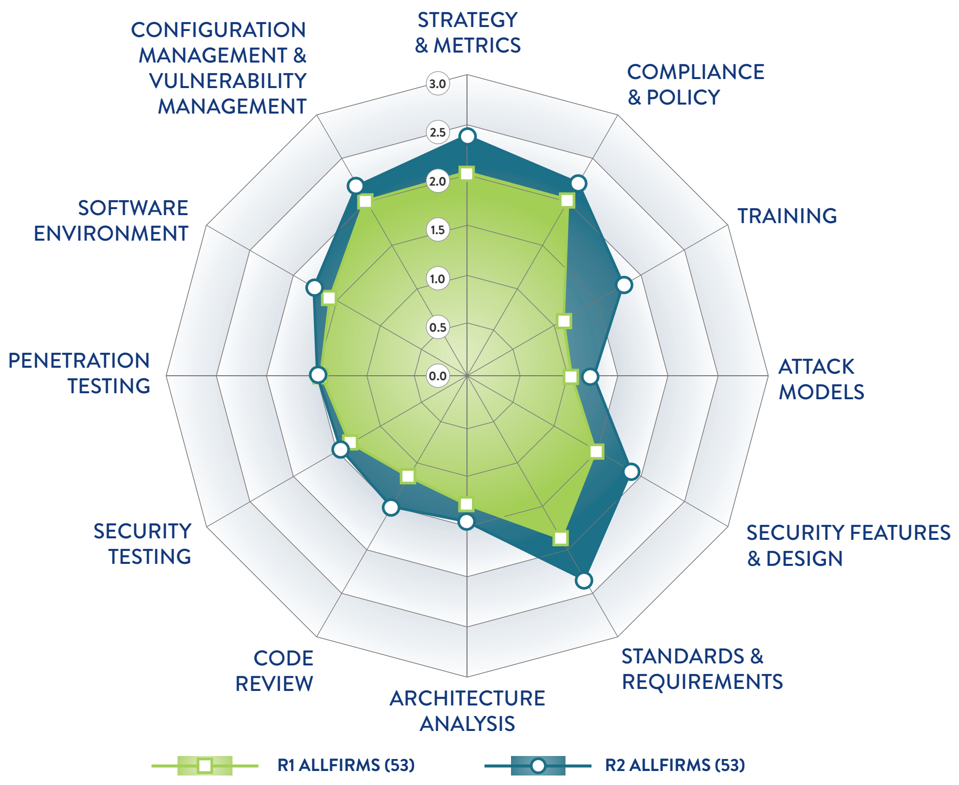


Figure 10. Round 1 Firms vs. Round 2 Firms Spider Chart. *This diagram illustrates the high-water mark change in 53 firms between their first and second BSIMM assessments.*

Changes to Longitudinal Scorecard

There are two obvious factors causing the numerical change seen on the longitudinal scorecard (showing 53 BSIMM11 firms moving from their first to second assessment). The first factor is newly observed activities. The activities where we see the biggest increase in new observations include the following:

- [SM1.1 Publish process and evolve as necessary], with 19 new observations.
- [SM1.2 Create evangelism role and perform internal marketing], with 19 new observations.
- [SM2.3 Create or grow a satellite], with 19 new observations.
- [SM2.1 Publish data about software security internally], with 18 new observations.
- [CP2.1 Build PII inventory], with 18 new observations.
- [SR2.2 Create a standards review board], with 18 new observations.
- [CMVM2.3 Develop an operations inventory of software delivery value streams], with 18 new observations.
- [CP1.3 Create policy], with 17 new observations.
- [PT1.2 Feed results to the defect management and mitigation system], with 17 new observations.

The changes observed from the first assessment to the second cannot always be seen directly on this scorecard, though. For example, [CR1.2 Perform opportunistic code review] was a new activity for 15 firms, but the scorecard shows that the observations increased by only four. That's where the second factor comes in: although that activity was newly observed in 15 firms, it was either no longer observed in 11 firms or went away due to data aging out, giving a total change of four (as shown in the scorecard). In a different example, the activity [SM1.2 Create evangelism role and perform internal marketing] was newly observed in 19 firms and dropped out of the data pool or was no longer observed in six firms. Therefore, the total observation count changed by 13 on the scorecard.

Twenty-one of the 130 BSIMM11 firms have had a third measurement. Table 7 captures the ongoing growth that occurs in these SSIs.

GOVERNANCE			INTELLIGENCE			SSDL TOUCHPOINTS			DEPLOYMENT		
ACTIVITY	BSIMM ROUND 1 (OF 21)	BSIMM ROUND 3 (OF 21)	ACTIVITY	BSIMM ROUND 1 (OF 21)	BSIMM ROUND 3 (OF 21)	ACTIVITY	BSIMM ROUND 1 (OF 21)	BSIMM ROUND 3 (OF 21)	ACTIVITY	BSIMM ROUND 1 (OF 21)	BSIMM ROUND 3 (OF 21)
[SM1.1]	10	19	[AM1.2]	14	20	[AA1.1]	18	20	[PT1.1]	19	19
[SM1.2]	8	15	[AM1.3]	5	15	[AA1.2]	6	9	[PT1.2]	11	21
[SM1.3]	10	16	[AM1.5]	11	11	[AA1.3]	5	8	[PT1.3]	11	15
[SM1.4]	18	21	[AM2.1]	4	7	[AA1.4]	11	14	[PT2.2]	5	5
[SM2.1]	6	15	[AM2.2]	2	6	[AA2.1]	3	5	[PT2.3]	10	4
[SM2.2]	6	14	[AM2.5]	3	5	[AA2.2]	0	5	[PT3.1]	2	2
[SM2.3]	9	12	[AM2.6]	2	1	[AA3.1]	3	2	[PT3.2]	1	2
[SM2.6]	10	12	[AM2.7]	2	4	[AA3.2]	0	0			
[SM3.1]	5	8	[AM3.1]	0	1	[AA3.3]	4	2			
[SM3.2]	0	4	[AM3.2]	0	2						
[SM3.3]	4	6	[AM3.3]	0	0						
[SM3.4]	0	0									
[CP1.1]	13	21	[SFD1.1]	18	19	[CR1.2]	10	17	[SE1.1]	11	17
[CP1.2]	18	20	[SFD1.2]	14	18	[CR1.4]	11	20	[SE1.2]	18	21
[CP1.3]	11	19	[SFD2.1]	4	10	[CR1.5]	5	7	[SE2.2]	7	4
[CP2.1]	10	13	[SFD2.2]	6	15	[CR1.6]	8	10	[SE2.4]	4	8
[CP2.2]	7	8	[SFD3.1]	1	7	[CR1.7]	2	12	[SE2.5]	0	2
[CP2.3]	10	14	[SFD3.2]	4	7	[CR2.6]	2	7	[SE2.6]	0	3
[CP2.4]	5	10	[SFD3.3]	2	0	[CR2.7]	4	7	[SE3.2]	0	3
[CP2.5]	7	13				[CR3.2]	0	1	[SE3.3]	5	2
[CP3.1]	4	6				[CR3.3]	0	0	[SE3.5]	0	2
[CP3.2]	4	2				[CR3.4]	0	0	[SE3.6]	0	1
[CP3.3]	1	2				[CR3.5]	0	1			
[T1.1]	14	20	[SR1.1]	14	19	[ST1.1]	13	18	[CMVM1.1]	19	21
[T1.5]	3	10	[SR1.2]	14	21	[ST1.3]	16	19	[CMVM1.2]	21	20
[T1.7]	9	15	[SR1.3]	13	20	[ST2.1]	7	11	[CMVM2.1]	19	20
[T1.8]	4	10	[SR2.2]	8	16	[ST2.4]	3	2	[CMVM2.2]	13	19
[T2.5]	3	7	[SR2.4]	4	12	[ST2.5]	0	1	[CMVM2.3]	13	17
[T2.8]	4	8	[SR2.5]	4	8	[ST2.6]	3	4	[CMVM3.1]	0	0
[T3.1]	0	2	[SR3.1]	2	7	[ST3.3]	1	1	[CMVM3.2]	1	2
[T3.2]	0	4	[SR3.2]	3	4	[ST3.4]	0	1	[CMVM3.3]	1	3
[T3.3]	0	3	[SR3.3]	6	4	[ST3.5]	2	1	[CMVM3.4]	0	5
[T3.4]	0	5	[SR3.4]	7	8	[ST3.6]	0	0	[CMVM3.5]	0	1
[T3.5]	0	3							[CMVM3.6]	0	0
[T3.6]	1	1									

Table 9. BSIMM11 Reassessments Scorecard Round 1 vs. Round 3. *The chart shows how 21 SSIs changed from their first to their third assessment.*

Figure 11 shows the average high-water marks per practice for the 21 firms in their first and third assessments. Interestingly, while this chart shows growth in nearly every practice, it also shows a slight regression in the Penetration Testing practice.

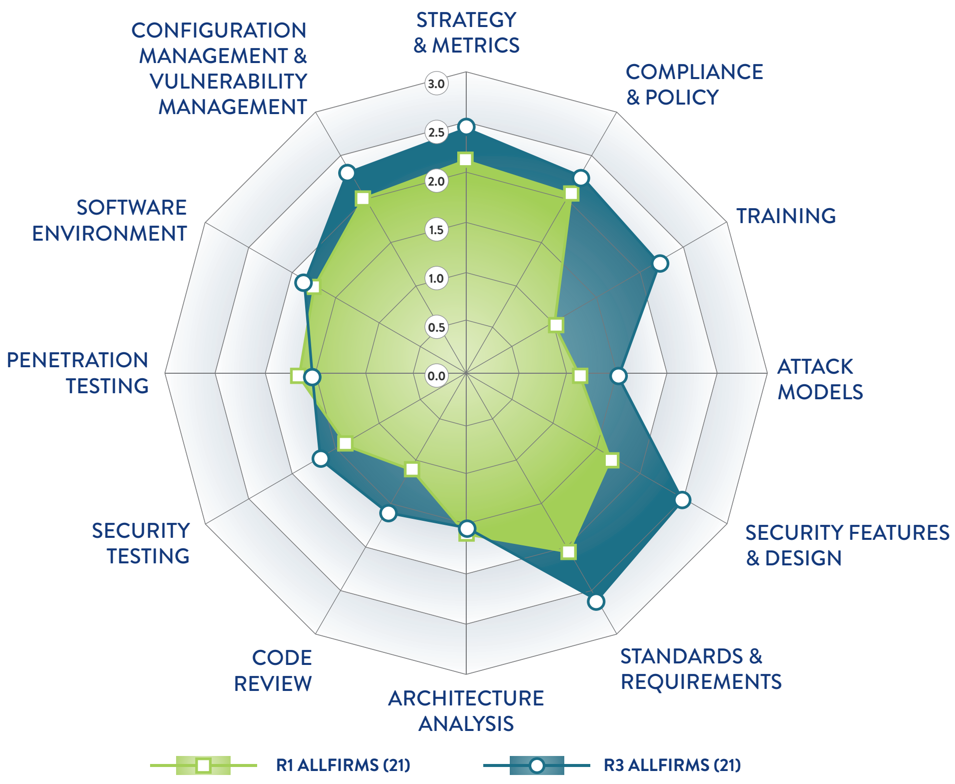


Figure 11. Round 1 Firms vs. Round 3 Firms Spider Chart. *This diagram illustrates the high-water mark change in 21 firms between their first and third BSIMM assessments.*

CHARTS, GRAPHS, AND SCORECARDS

In this section, we present the BSIMM skeleton showing the activities and their observation rates, along with other useful or interesting charts.

The BSIMM skeleton provides a way to view the model at a glance and is useful when assessing an SSI. We showed a streamlined version of the skeleton in Part Two. Table 10, below, shows a more detailed version, with the number and percentages of firms (out of 130) performing that activity in their own SSI.

GOVERNANCE

STRATEGY & METRICS (SM)

ACTIVITY DESCRIPTION	ACTIVITY	OBSERVATIONS	PARTICIPANT %
LEVEL 1			
Publish process and evolve as necessary.	[SM1.1]	94	72.3%
Create evangelism role and perform internal marketing.	[SM1.2]	70	53.8%
Educate executives.	[SM1.3]	75	57.7%
Implement lifecycle governance.	[SM1.4]	117	90.0%
LEVEL 2			
Publish data about software security internally.	[SM2.1]	64	49.2%
Verify release conditions with measurements and track exceptions.	[SM2.2]	61	46.9%
Create or grow a satellite.	[SM2.3]	55	42.3%
Require security sign-off.	[SM2.6]	65	50.0%
LEVEL 3			
Use a software asset tracking application with portfolio view.	[SM3.1]	27	20.8%
Run an external marketing program.	[SM3.2]	4	3.1%
Identify metrics and use them to drive resourcing.	[SM3.3]	18	13.8%
Integrate software-defined lifecycle governance.	[SM3.4]	1	0.8%

COMPLIANCE & POLICY (CP)

ACTIVITY DESCRIPTION	ACTIVITY	OBSERVATIONS	PARTICIPANT %
LEVEL 1			
Unify regulatory pressures.	[CP1.1]	94	72.3%
Identify PII obligations.	[CP1.2]	112	86.2%
Create policy.	[CP1.3]	86	66.2%
LEVEL 2			
Build PII inventory.	[CP2.1]	55	42.3%
Require security sign-off for compliance-related risk.	[CP2.2]	47	36.2%
Implement and track controls for compliance.	[CP2.3]	61	46.9%
Include software security SLAs in all vendor contracts.	[CP2.4]	48	36.9%
Ensure executive awareness of compliance and privacy obligations.	[CP2.5]	70	53.8%
LEVEL 3			
Create a regulator compliance story.	[CP3.1]	26	20.0%
Impose policy on vendors.	[CP3.2]	16	12.3%
Drive feedback from software lifecycle data back to policy.	[CP3.3]	8	6.2%

TRAINING (T)

ACTIVITY DESCRIPTION	ACTIVITY	OBSERVATIONS	PARTICIPANT %
LEVEL 1			
Conduct awareness training.	[T1.1]	83	63.8%
Deliver role-specific advanced curriculum.	[T1.5]	38	29.2%
Deliver on-demand individual training.	[T1.7]	53	40.8%
Include security resources in onboarding.	[T1.8]	41	31.5%
LEVEL 2			
Enhance satellite through training and events.	[T2.5]	32	24.6%
Create and use material specific to company history.	[T2.8]	28	21.5%
LEVEL 3			
Reward progression through curriculum.	[T3.1]	4	3.1%
Provide training for vendors and outsourced workers.	[T3.2]	22	16.9%
Host software security events.	[T3.3]	16	12.3%
Require an annual refresher.	[T3.4]	19	14.6%
Establish SSG office hours.	[T3.5]	7	5.4%
Identify new satellite members through observation.	[T3.6]	1	0.8%

INTELLIGENCE

ATTACK MODELS (AM)

ACTIVITY DESCRIPTION	ACTIVITY	OBSERVATIONS	PARTICIPANT %
LEVEL 1			
Create a data classification scheme and inventory.	[AM1.2]	81	62.3%
Identify potential attackers.	[AM1.3]	38	29.2%
Gather and use attack intelligence.	[AM1.5]	57	43.8%
LEVEL 2			
Build attack patterns and abuse cases tied to potential attackers.	[AM2.1]	12	9.2%
Create technology-specific attack patterns.	[AM2.2]	10	7.7%
Build and maintain a top N possible attacks list.	[AM2.5]	16	12.3%
Collect and publish attack stories.	[AM2.6]	10	7.7%
Build an internal forum to discuss attacks.	[AM2.7]	14	10.8%
LEVEL 3			
Have a research group that develops new attack methods.	[AM3.1]	3	2.3%
Create and use automation to mimic attackers.	[AM3.2]	4	3.1%
Monitor automated asset creation.	[AM3.3]	4	3.1%

SECURITY FEATURES & DESIGN (SFD)

ACTIVITY DESCRIPTION	ACTIVITY	OBSERVATIONS	PARTICIPANT %
LEVEL 1			
Integrate and deliver security features.	[SFD1.1]	102	78.5%
Engage the SSG with architecture teams.	[SFD1.2]	76	58.5%
LEVEL 2			
Leverage secure-by-design components and services.	[SFD2.1]	32	24.6%
Create capability to solve difficult design problems.	[SFD2.2]	51	39.2%
LEVEL 3			
Form a review board or central committee to approve and maintain secure design patterns.	[SFD3.1]	14	10.8%
Require use of approved security features and frameworks.	[SFD3.2]	14	10.8%
Find and publish secure design patterns from the organization.	[SFD3.3]	4	3.1%

STANDARDS & REQUIREMENTS (SR)

ACTIVITY DESCRIPTION	ACTIVITY	OBSERVATIONS	PARTICIPANT %
LEVEL 1			
Create security standards.	[SR1.1]	93	71.5%
Create a security portal.	[SR1.2]	90	69.2%
Translate compliance constraints to requirements.	[SR1.3]	94	72.3%
LEVEL 2			
Create a standards review board.	[SR2.2]	64	49.2%
Identify open source.	[SR2.4]	60	46.2%
Create SLA boilerplate.	[SR2.5]	44	33.8%
LEVEL 3			
Control open source risk.	[SR3.1]	30	23.1%
Communicate standards to vendors.	[SR3.2]	9	6.9%
Use secure coding standards.	[SR3.3]	9	6.9%
Create standards for technology stacks.	[SR3.4]	25	19.2%

SSDL TOUCHPOINTS

ARCHITECTURE ANALYSIS (AA)

ACTIVITY DESCRIPTION	ACTIVITY	OBSERVATIONS	PARTICIPANT %
LEVEL 1			
Perform security feature review.	[AA1.1]	114	87.7%
Perform design review for high-risk applications.	[AA1.2]	41	31.5%
Have SSG lead design review efforts.	[AA1.3]	32	24.6%
Use a risk methodology to rank applications.	[AA1.4]	67	51.5%
LEVEL 2			
Define and use AA process.	[AA2.1]	23	17.7%
Standardize architectural descriptions.	[AA2.2]	24	18.5%
LEVEL 3			
Have engineering teams lead AA process.	[AA3.1]	11	8.5%
Drive analysis results into standard architecture patterns.	[AA3.2]	1	0.8%
Make the SSG available as an AA resource or mentor.	[AA3.3]	6	4.6%

CODE REVIEW (CR)

ACTIVITY DESCRIPTION	ACTIVITY	OBSERVATIONS	PARTICIPANT %
LEVEL 1			
Perform opportunistic code review.	[CR1.2]	79	60.8%
Use automated tools along with manual review.	[CR1.4]	100	76.9%
Make code review mandatory for all projects.	[CR1.5]	49	37.7%
Use centralized reporting to close the knowledge loop and drive training.	[CR1.6]	41	31.5%
Assign tool mentors.	[CR1.7]	50	38.5%
LEVEL 2			
Use automated tools with tailored rules.	[CR2.6]	23	17.7%
Use a top N bugs list (real data preferred).	[CR2.7]	24	18.5%
LEVEL 3			
Build a capability to combine assessment results.	[CR3.2]	7	5.4%
Create a capability to eradicate bugs.	[CR3.3]	3	2.3%
Automate malicious code detection.	[CR3.4]	2	1.5%
Enforce coding standards.	[CR3.5]	1	0.8%

SECURITY TESTING (ST)			
ACTIVITY DESCRIPTION	ACTIVITY	OBSERVATIONS	PARTICIPANT %
LEVEL 1			
Ensure QA performs edge/boundary value condition testing.	[ST1.1]	104	80.0%
Drive tests with security requirements and security features.	[ST1.3]	89	68.5%
LEVEL 2			
Integrate black-box security tools into the QA process.	[ST2.1]	42	32.3%
Share security results with QA.	[ST2.4]	20	15.4%
Include security tests in QA automation.	[ST2.5]	16	12.3%
Perform fuzz testing customized to application APIs.	[ST2.6]	13	10%
LEVEL 3			
Drive tests with risk analysis results.	[ST3.3]	5	3.8%
Leverage coverage analysis.	[ST3.4]	2	1.5%
Begin to build and apply adversarial security tests (abuse cases).	[ST3.5]	2	1.5%
Implement event-driven security testing in automation.	[ST3.6]	0	0.0

DEPLOYMENT

PENETRATION TESTING (PT)

ACTIVITY DESCRIPTION	ACTIVITY	OBSERVATIONS	PARTICIPANT %
LEVEL 1			
Use external penetration testers to find problems.	[PT1.1]	114	87.7%
Feed results to the defect management and mitigation system.	[PT1.2]	100	76.9%
Use penetration testing tools internally.	[PT1.3]	89	68.5%
LEVEL 2			
Penetration testers use all available information.	[PT2.2]	30	23.1%
Schedule periodic penetration tests for application coverage.	[PT2.3]	29	22.3%
LEVEL 3			
Use external penetration testers to perform deep-dive analysis.	[PT3.1]	21	16.2%
Customize penetration testing tools and scripts.	[PT3.2]	10	7.7%

SOFTWARE ENVIRONMENT (SE)

ACTIVITY DESCRIPTION	ACTIVITY	OBSERVATIONS	PARTICIPANT %
LEVEL 1			
Use application input monitoring.	[SE1.1]	73	56.2%
Ensure host and network security basics are in place.	[SE1.2]	121	93.1%
LEVEL 2			
Define secure deployment parameters and configurations.	[SE2.2]	39	30%
Protect code integrity.	[SE2.4]	33	25.4%
Use application containers.	[SE2.5]	31	23.8%
Ensure cloud security basics.	[SE2.6]	36	27.7%
LEVEL 3			
Use code protection.	[SE3.2]	13	10.0%
Use application behavior monitoring and diagnostics.	[SE3.3]	7	5.4%
Use orchestration for containers and virtualized environments.	[SE3.5]	22	16.9%
Enhance application inventory with operations bill of materials.	[SE3.6]	12	9.2%

CONFIGURATION MANAGEMENT & VULNERABILITY MANAGEMENT (CMVM)			
ACTIVITY DESCRIPTION	ACTIVITY	OBSERVATIONS	PARTICIPANT %
LEVEL 1			
Create or interface with incident response.	[CMVM1.1]	108	83.1%
Identify software defects found in operations monitoring and feed them back to development.	[CMVM1.2]	102	78.5%
LEVEL 2			
Have emergency response.	[CMVM2.1]	94	72.3%
Track software bugs found in operations through the fix process.	[CMVM2.2]	97	74.6%
Develop an operations inventory of software delivery value streams.	[CMVM2.3]	65	50.0%
LEVEL 3			
Fix all occurrences of software bugs found in operations.	[CMVM3.1]	5	3.8%
Enhance the SSDL to prevent software bugs found in operations.	[CMVM3.2]	11	8.5%
Simulate software crises.	[CMVM3.3]	13	10.0%
Operate a bug bounty program.	[CMVM3.4]	16	12.3%
Automate verification of operational infrastructure security.	[CMVM3.5]	8	6.2%
Publish risk data for deployable artifacts.	[CMVM3.6]	0	0.0

Table 10. BSIMM11 Skeleton. *This expanded version of the BSIMM skeleton shows the 12 BSIMM practices and the 121 activities they contain, along with the observation rates as both counts and percentages. Highlighted activities are the most common per practice.*

Figure 12 shows the distribution of scores among the population of 130 participating firms. To create this graph, we divided the scores into six bins. 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 the horizontal line on the graph. In general, firms where more BSIMM activities have been observed have older SSIs.

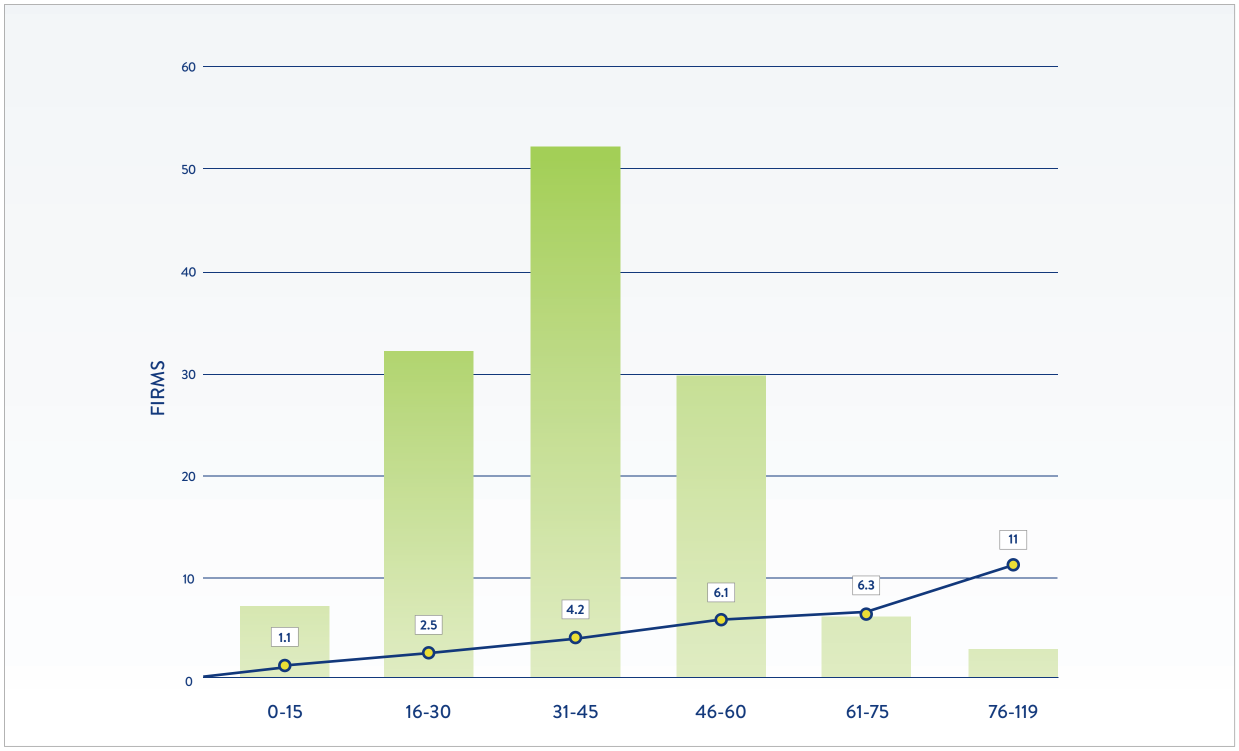


Figure 12. BSIMM Score Distribution. *The majority of BSIMM11 participants have a score in the 16 to 45 range, with an average SSG age of 2.5 to 4.2 years.*

COMPARING VERTICALS

Table 11 shows the BSIMM scorecards for the nine verticals compared side by side. In the Activity columns, we have highlighted the most common activity in each practice as observed in the entire BSIMM data pool (130 firms). See Part Two for discussion.

GOVERNANCE									
ACTIVITY	FINANCIAL (OF 42)	FINTECH (OF 21)	ISV (OF 46)	TECH (OF 27)	HEALTHCARE (OF 14)	IOT (OF 17)	INSURANCE (OF 14)	CLOUD (OF 30)	RETAIL (OF 8)
[SM1.1]	29	18	35	22	11	14	9	23	5
[SM1.2]	19	11	27	20	8	12	3	15	5
[SM1.3]	25	11	28	17	8	10	6	19	4
[SM1.4]	39	20	40	26	13	15	12	25	8
[SM2.1]	27	15	20	9	6	7	6	19	5
[SM2.2]	24	11	20	15	6	9	3	13	3
[SM2.3]	15	9	22	14	6	10	6	13	4
[SM2.6]	25	8	19	18	8	11	5	13	4
[SM3.1]	11	6	7	6	2	4	2	6	1
[SM3.2]	0	1	4	2	0	1	0	2	1
[SM3.3]	11	2	3	2	3	2	3	2	0
[SM3.4]	1	1	0	0	0	0	0	0	0
[CP1.1]	31	18	32	20	14	14	10	21	4
[CP1.2]	39	20	34	22	14	15	13	26	8
[CP1.3]	32	17	27	16	8	11	10	19	4
[CP2.1]	22	10	16	9	8	10	3	14	4
[CP2.2]	18	5	13	14	9	9	4	8	3
[CP2.3]	22	10	20	15	11	8	5	14	2
[CP2.4]	16	7	18	10	6	6	6	12	4
[CP2.5]	22	15	25	14	9	10	6	21	2
[CP3.1]	13	6	8	2	2	2	2	7	1
[CP3.2]	7	3	2	3	3	1	3	2	1
[CP3.3]	4	1	2	2	1	1	1	2	0
[T1.1]	27	19	32	18	5	14	4	22	6
[T1.5]	13	6	15	10	3	6	4	9	4
[T1.7]	18	11	19	10	3	8	5	15	4
[T1.8]	15	10	17	8	1	6	4	13	2
[T2.5]	9	6	11	7	1	5	3	7	3
[T2.8]	5	2	15	10	2	8	1	10	2
[T3.1]	1	0	2	2	0	1	1	2	0
[T3.2]	6	6	10	7	2	6	2	9	1
[T3.3]	7	2	8	5	0	3	0	3	0
[T3.4]	10	4	5	2	2	2	4	4	1
[T3.5]	3	1	2	2	0	0	1	1	1
[T3.6]	0	0	1	0	0	0	0	1	0

Table 11. Vertical Comparison Scorecard. This table allows for easy comparisons of observation rates for the nine verticals tracked in BSIMM11.

INTELLIGENCE									
ACTIVITY	FINANCIAL (OF 42)	FINTECH (OF 21)	ISV (OF 46)	TECH (OF 27)	HEALTHCARE (OF 14)	IOT (OF 17)	INSURANCE (OF 14)	CLOUD (OF 30)	RETAIL (OF 8)
[AM1.2]	37	14	21	9	11	8	10	16	6
[AM1.3]	17	4	8	8	6	5	7	5	1
[AM1.5]	21	13	14	12	8	7	6	12	4
[AM2.1]	5	2	2	4	2	3	3	1	1
[AM2.2]	4	2	4	5	0	3	0	2	0
[AM2.5]	3	3	7	8	2	5	1	4	1
[AM2.6]	2	1	5	4	2	4	1	4	0
[AM2.7]	3	1	6	7	2	4	1	3	0
[AM3.1]	0	1	1	2	0	1	0	1	1
[AM3.2]	0	0	2	3	0	1	0	2	0
[AM3.3]	2	1	1	0	0	0	0	2	0
[SFD1.1]	34	17	33	18	12	10	11	25	8
[SFD1.2]	18	15	32	20	10	12	7	22	6
[SFD2.1]	10	6	14	10	2	6	1	8	1
[SFD2.2]	12	8	22	13	4	8	3	15	4
[SFD3.1]	10	2	1	1	1	0	1	0	1
[SFD3.2]	4	2	5	2	1	1	2	6	1
[SFD3.3]	0	0	2	2	1	2	0	0	1
[SR1.1]	34	18	26	18	10	13	10	18	6
[SR1.2]	27	16	38	20	9	14	7	25	5
[SR1.3]	29	18	33	21	11	13	7	22	7
[SR2.2]	29	11	14	11	4	5	9	13	5
[SR2.4]	21	13	25	13	5	9	7	15	2
[SR2.5]	15	5	15	11	6	8	5	10	4
[SR3.1]	11	8	12	7	1	4	2	8	1
[SR3.2]	2	1	2	3	2	1	2	2	0
[SR3.3]	2	2	2	3	1	2	1	2	0
[SR3.4]	11	4	8	4	2	4	1	7	2

Table 11. Vertical Comparison Scorecard. This table allows for easy comparisons of observation rates for the nine verticals tracked in BSIMM11.

	SSDL TOUCHPOINTS								
ACTIVITY	FINANCIAL (OF 42)	FINTECH (OF 21)	ISV (OF 46)	TECH (OF 27)	HEALTHCARE (OF 14)	IOT (OF 17)	INSURANCE (OF 14)	CLOUD (OF 30)	RETAIL (OF 8)
[AA1.1]	36	21	40	24	13	15	11	27	8
[AA1.2]	11	5	16	16	3	9	2	9	1
[AA1.3]	9	3	12	11	4	5	3	7	1
[AA1.4]	32	14	14	7	11	6	11	9	7
[AA2.1]	5	0	11	14	3	6	2	5	0
[AA2.2]	6	1	10	12	3	7	3	6	0
[AA3.1]	3	0	4	8	0	6	1	2	0
[AA3.2]	0	0	0	0	0	0	0	0	1
[AA3.3]	1	0	2	5	0	3	0	2	0
[CR1.2]	26	10	26	19	8	10	8	16	4
[CR1.4]	33	19	36	19	8	13	9	25	6
[CR1.5]	13	10	19	12	5	6	4	9	2
[CR1.6]	15	8	17	6	3	4	1	11	4
[CR1.7]	18	10	19	9	3	5	6	12	5
[CR2.6]	11	7	4	2	1	1	1	6	1
[CR2.7]	9	4	7	4	1	2	4	6	1
[CR3.2]	1	1	1	2	2	1	2	1	1
[CR3.3]	1	1	1	0	0	0	0	2	0
[CR3.4]	1	1	0	0	0	0	0	0	0
[CR3.5]	1	0	0	0	0	0	0	0	0
[ST1.1]	34	19	38	25	8	16	11	24	6
[ST1.3]	30	16	33	20	8	13	10	19	5
[ST2.1]	11	9	19	13	4	10	5	8	4
[ST2.4]	3	3	8	10	2	6	2	5	1
[ST2.5]	5	4	6	7	1	4	1	5	1
[ST2.6]	0	1	9	10	1	6	0	3	0
[ST3.3]	0	0	3	4	0	4	0	3	0
[ST3.4]	1	0	0	1	1	1	1	0	0
[ST3.5]	0	0	2	1	0	1	0	2	0
[ST3.6]	0	0	0	0	0	0	0	0	0

Table 11. Vertical Comparison Scorecard. *This table allows for easy comparisons of observation rates for the nine verticals tracked in BSIMM11.*

DEPLOYMENT									
ACTIVITY	FINANCIAL (OF 42)	FINTECH (OF 21)	ISV (OF 46)	TECH (OF 27)	HEALTHCARE (OF 14)	IOT (OF 17)	INSURANCE (OF 14)	CLOUD (OF 30)	RETAIL (OF 8)
[PT1.1]	36	20	41	24	13	14	13	23	8
[PT1.2]	33	20	35	19	9	12	8	22	8
[PT1.3]	32	15	29	19	9	9	9	20	7
[PT2.2]	6	3	13	11	4	7	3	10	1
[PT2.3]	13	5	11	4	1	1	2	7	3
[PT3.1]	3	4	8	12	3	7	1	5	2
[PT3.2]	4	3	4	4	0	1	0	3	0
[SE1.1]	33	13	17	9	10	8	10	16	4
[SE1.2]	41	20	42	25	13	16	12	28	7
[SE2.2]	12	4	16	15	1	10	3	9	2
[SE2.4]	3	5	20	17	1	12	1	9	1
[SE2.5]	7	6	14	7	3	2	3	14	3
[SE2.6]	13	4	14	5	3	3	4	16	1
[SE3.2]	3	1	5	6	2	3	2	2	2
[SE3.3]	3	1	4	1	1	1	1	1	1
[SE3.5]	6	2	11	3	2	1	3	12	1
[SE3.6]	4	2	6	4	0	4	0	4	0
[CMVM1.1]	37	20	39	22	10	14	12	25	7
[CMVM1.2]	34	17	35	21	10	16	9	23	8
[CMVM2.1]	33	17	33	19	10	13	8	22	7
[CMVM2.2]	32	17	34	21	8	14	8	21	8
[CMVM2.3]	29	10	22	9	6	7	6	14	3
[CMVM3.1]	2	2	1	1	0	1	0	1	0
[CMVM3.2]	3	1	5	5	2	3	1	4	0
[CMVM3.3]	3	3	3	4	2	2	1	2	1
[CMVM3.4]	5	3	6	2	1	1	3	7	1
[CMVM3.5]	4	2	2	1	0	1	1	4	0
[CMVM3.6]	0	0	0	0	0	0	0	0	0

Table 11. Vertical Comparison Scorecard. This table allows for easy comparisons of observation rates for the nine verticals tracked in BSIMM11.

121 BSIMM ACTIVITIES AT A GLANCE

Included below is a list of BSIMM11 activities sorted into their respective levels. Keep in mind that the levels in the BSIMM are simply an organizing feature that allows everyone to quickly determine the relative frequency with which each activity is observed. Those observed frequently will be at level 1, while those observed very infrequently will be at level 3. Some activities will be inherently harder to implement for some organizations (e.g., a large organization with thousands of applications versus a small organization with a handful), but that isn't what's reflected by the model's levels.

Unlike other maturity models, each BSIMM activity is unique. A model with multiple prescriptive yet vague variations of activities (e.g., do X, do more of X, formalize X, make X repeatable) is not a viable substitute for an observational model of things organizations actually do. In our anecdotal experience, organizations that measure their efforts via the procedural formality of each activity instead of the depth, breadth, and cost-effectiveness of their overall SSI are only seeing the must-report-progress trees, not the software security forest.

The BSIMM focuses on the real-world "what" of software security, leaving the "how" and the "how much" to each organization and its unique culture, budget, resources, constraints, software portfolio, and business objectives. Generally, the "how" is trending toward automation, so it's time to integrate an "automate first" strategy into your SSI, where the "how much" will derive from consistently applying a clearly defined risk management approach that is connected to other organizational cybersecurity risk management initiatives (e.g., cloud, privacy, digital transformation).

LEVEL 1 ACTIVITIES

*(**Blue** indicates most observed BSIMM activity in that practice.)*

Governance

Strategy & Metrics (SM)

- Publish process and evolve as necessary. [SM1.1]
- Create evangelism role and perform internal marketing. [SM1.2]
- Educate executives. [SM1.3]
- **Implement lifecycle governance. [SM1.4]**

Compliance & Policy (CP)

- Unify regulatory pressures. [CP1.1]
- **Identify PII obligations. [CP1.2]**
- Create policy. [CP1.3]

Training (T)

- **Conduct awareness training. [T1.1]**
- Deliver role-specific advanced curriculum. [T1.5]
- Deliver on-demand individual training. [T1.7]
- Include security resources in onboarding. [T1.8]

Intelligence

Attack Models (AM)

- **Create a data classification scheme and inventory. [AM1.2]**
- Identify potential attackers. [AM1.3]
- Gather and use attack intelligence. [AM1.5]

Security Features & Design (SFD)

- **Integrate and deliver security features. [SFD1.1]**
- Engage the SSG with architecture teams. [SFD1.2]

Standards & Requirements (SR)

- Create security standards. [SR1.1]
- Create a security portal. [SR1.2]
- **Translate compliance constraints to requirements. [SR1.3]**

SSDL Touchpoints

Architecture Analysis (AA)

- **Perform security feature review. [AA1.1]**
- Perform design review for high-risk applications. [AA1.2]
- Have SSG lead design review efforts. [AA1.3]
- Use a risk methodology to rank applications. [AA1.4]

Code Review (CR)

- Perform opportunistic code review. [CR1.2]
- **Use automated tools along with manual review. [CR1.4]**
- Make code review mandatory for all projects. [CR1.5]
- Use centralized reporting to close the knowledge loop and drive training. [CR1.6]
- Assign tool mentors. [CR1.7]

Security Testing (ST)

- **Ensure QA performs edge/boundary value condition testing. [ST1.1]**
- Drive tests with security requirements and security features. [ST1.3]

Deployment

Penetration Testing (PT)

- **Use external penetration testers to find problems. [PT1.1]**
- Feed results to the defect management and mitigation system. [PT1.2]
- Use penetration testing tools internally. [PT1.3]

Software Environment (SE)

- Use application input monitoring. [SE1.1]
- **Ensure host and network security basics are in place. [SE1.2]**

Configuration Management & Vulnerability Management (CMVM)

- **Create or interface with incident response. [CMVM1.1]**
- Identify software defects found in operations monitoring and feed them back to development. [CMVM1.2]

LEVEL 2 ACTIVITIES

Governance

Strategy & Metrics (SM)

- Publish data about software security internally. [SM2.1]
- Verify release conditions with measurements and track exceptions. [SM2.2]
- Create or grow a satellite. [SM2.3]
- Require security sign-off. [SM2.6]

Compliance & Policy (CP)

- Build PII inventory. [CP2.1]
- Require security sign-off for compliance-related risk. [CP2.2]
- Implement and track controls for compliance. [CP2.3]
- Include software security SLAs in all vendor contracts. [CP2.4]
- Ensure executive awareness of compliance and privacy obligations. [CP2.5]

Training (T)

- Enhance satellite through training and events. [T2.5]
- Create and use material specific to company history. [T2.8]

Intelligence

Attack Models (AM)

- Build attack patterns and abuse cases tied to potential attackers. [AM2.1]
- Create technology-specific attack patterns. [AM2.2]
- Build and maintain a top N possible attacks list. [AM2.5]
- Collect and publish attack stories. [AM2.6]
- Build an internal forum to discuss attacks. [AM2.7]

Security Features & Design (SFD)

- Leverage secure-by-design components and services. [SFD2.1]
- Create capability to solve difficult design problems. [SFD2.2]

Standards & Requirements (SR)

- Create a standards review board. [SR2.2]
- Identify open source. [SR2.4]
- Create SLA boilerplate. [SR2.5]

SSDL Touchpoints

Architecture Analysis (AA)

- Define and use AA process. [AA2.1]
- Standardize architectural descriptions. [AA2.2]

Code Review (CR)

- Use automated tools with tailored rules. [CR2.6]
- Use a top N bugs list (real data preferred). [CR2.7]

Security Testing (ST)

- Integrate black-box security tools into the QA process. [ST2.1]
- Share security results with QA. [ST2.4]
- Include security tests in QA automation. [ST2.5]
- Perform fuzz testing customized to application APIs. [ST2.6]

Deployment

Penetration Testing (PT)

- Penetration testers use all available information. [PT2.2]
- Schedule periodic penetration tests for application coverage. [PT2.3]

Software Environment (SE)

- Define secure deployment parameters and configurations. [SE2.2]
- Protect code integrity. [SE2.4]
- Use application containers. [SE2.5]
- Ensure cloud security basics. [SE2.6]

Configuration Management & Vulnerability Management (CMVM)

- Have emergency response. [CMVM2.1]
- Track software bugs found in operations through the fix process. [CMVM2.2]
- Develop an operations inventory of software delivery value streams. [CMVM2.3]

LEVEL 3 ACTIVITIES

Governance

Strategy & Metrics (SM)

- Use a software asset tracking application with portfolio view. [SM3.1]
- Run an external marketing program. [SM3.2]
- Identify metrics and use them to drive resourcing. [SM3.3]
- Integrate software-defined lifecycle governance. [SM3.4]

Compliance & Policy (CP)

- Create a regulator compliance story. [CP3.1]
- Impose policy on vendors. [CP3.2]
- Drive feedback from software lifecycle data back to policy. [CP3.3]

Training (T)

- Reward progression through curriculum. [T3.1]
- Provide training for vendors and outsourced workers. [T3.2]
- Host software security events. [T3.3]
- Require an annual refresher. [T3.4]
- Establish SSG office hours. [T3.5]
- Identify new satellite members through observation. [T3.6]

Intelligence

Attack Models (AM)

- Have a research group that develops new attack methods. [AM3.1]
- Create and use automation to mimic attackers. [AM3.2]
- Monitor automated asset creation. [AM3.3]

Security Features & Design (SFD)

- Form a review board or central committee to approve and maintain secure design patterns. [SFD 3.1]
- Require use of approved security features and frameworks. [SFD3.2]
- Find and publish secure design patterns from the organization. [SFD3.3]

Standards & Requirements (SR)

- Control open source risk. [SR3.1]
- Communicate standards to vendors. [SR3.2]
- Use secure coding standards. [SR3.3]
- Create standards for technology stacks. [SR3.4]

SSDL Touchpoints

Architecture Analysis (AA)

- Have engineering teams lead AA process. [AA3.1]
- Drive analysis results into standard architecture patterns. [AA3.2]
- Make the SSG available as an AA resource or mentor. [AA3.3]

Code Review (CR)

- Build a capability to combine assessment results. [CR3.2]
- Create capability to eradicate bugs. [CR3.3]
- Automate malicious code detection. [CR3.4]
- Enforce coding standards. [CR3.5]

Security Testing (ST)

- Drive tests with risk analysis results. [ST3.3]
- Leverage coverage analysis. [ST3.4]
- Begin to build and apply adversarial security tests (abuse cases). [ST3.5]
- Implement event-driven security testing in automation. [ST3.6]

Deployment

Penetration Testing (PT)

- Use external penetration testers to perform deep-dive analysis. [PT3.1]
- Customize penetration testing tools. [PT3.2]

Software Environment (SE)

- Use code protection. [SE3.2]
- Use application behavior monitoring and diagnostics. [SE3.3]
- Use orchestration for containers and virtualized environments. [SE3.5]
- Enhance application inventory with operations bill of materials. [SE3.6]

Configuration Management & Vulnerability Management (CMVM)

- Fix all occurrences of software bugs found in operations. [CMVM3.1]
- Enhance the SSDL to prevent software bugs found in operations. [CMVM3.2]
- Simulate software crises. [CMVM3.3]
- Operate a bug bounty program. [CMVM3.4]
- Automate verification of operational infrastructure security. [CMVM3.5]
- Publish risk data for deployable artifacts. [CMVM3.6]

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 useful guidance, reflection upon past changes can help clarify the ebb and flow of particular activities. Table 12 shows the activity moves and adds that have occurred since the BSIMM's creation.

ACTIVITY CHANGES OVER TIME	
	CHANGED IN THE MODEL FOR BSIMM11 (121 ACTIVITIES)
SEPTEMBER 2020	• T2.6 Include security resources in onboarding became T1.8. • CR2.5 Assign tool mentors became CR1.7. • SE3.4 Use application containers 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.
	CHANGED IN THE MODEL FOR BSIMM10 (119 ACTIVITIES)
SEPTEMBER 2019	• T1.6 Create and use material specific to company history became T2.8. • SR2.3 Create standards for technology stacks became 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.
	CHANGED IN THE MODEL FOR BSIMM9 (116 ACTIVITIES)
OCTOBER 2018	• 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.
	CHANGED IN THE MODEL FOR BSIMM8 (113 ACTIVITIES)
SEPTEMBER 2017	• T2.7 Identify new satellite through observation became T3.6. • AA2.3 Make SSG available as an AA resource or mentor became AA3.3.
	CHANGED IN THE MODEL FOR BSIMM7 (113 ACTIVITIES)
OCTOBER 2016	• AM1.1 Build and maintain 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 (real data preferred) CR2.7. • CR2.2 Enforce coding standards became CR3.5. • SE3.4 Use application containers added to model.
	CHANGED IN THE MODEL FOR BSIMM6 (112 ACTIVITIES)
OCTOBER 2015	• SM1.6 Require security sign-off 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.
	CHANGED IN THE MODEL FOR BSIMM-V (112 ACTIVITIES)
OCTOBER 2013	• SFD2.3 Find and publish secure design patterns from the organization became SFD3.3. • SR2.1 Communicate standards to vendors became SR3.2. • CR3.1 Use automated tools with tailored rules became CR2.6. • ST2.3 Begin to build and apply adversarial security tests (abuse cases) became ST3.5. • CMVM3.4 Operate a bug bounty program added to model.

ACTIVITY CHANGES OVER TIME (continued)	
	CHANGED IN THE MODEL FOR BSIMM4 (111 ACTIVITIES)
SEPTEMBER 2012	• 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 and drive training became CR1.6. • ST1.2 Share security results with QA became ST2.4. • SE2.3 Use application behavior monitoring and diagnostics became SE3.3. • CR3.4 Automate malicious code detection added to model. • CMVM3.3 Simulate software crises added to model.
	CHANGED IN THE MODEL FOR BSIMM3 (109 ACTIVITIES)
SEPTEMBER 2011	• SM1.5 Identify metrics and use them to drive budgets 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 penetration testing tools internally became PT1.3.
	CHANGED IN THE MODEL FOR BSIMM2 (109 ACTIVITIES)
MAY 2010	• T2.3 Require an annual refresher became T3.4. • CR2.1 Use automated tools along with manual review 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.
	CHANGED IN THE MODEL FOR BSIMM1 110 ACTIVITIES)
MARCH 2009	• Added 110 activities.

Table 12. Activity Changes Over Time. *This table allows for historical review of how BSIMM activities have been added or moved over time.*

BSIMM Online Community

Find out how to unlock access to an engaged BSIMM member community, including conferences, newsletters, and original content.

www.bsimm.com