Gary McGraw, Ph.D., Sammy Migues, and Jacob West

Executive Summary

The Building Security in Maturity Model (BSIMM) is the result of a multiyear study of real-world software security initiatives. We present the BSIMM9 model as built directly out of data observed in 120 firms. Seventy of the firms are listed in the Acknowledgments section on page 3.

The BSIMM is a measuring stick for software security. The best way to use the BSIMM is to compare and contrast your own initiative with the data about what other organizations are doing contained in the model. You can then identify your own goals and objectives and refer to the BSIMM to determine which additional activities make sense for you.

The BSIMM data show that high maturity initiatives are well-rounded, carrying out numerous activities in all 12 of the practices described by the model. The model also describes how mature software security initiatives evolve, change, and improve over time.

BSIMM9 License

Acknowledgments

Our thanks to the 120 executives from the world-class software security initiatives we studied from around the world to create BSIMM9, including those who choose to remain anonymous.

Adobe
Aetna
Alibaba
Amgen
ANDA
Autodesk
Axway
Bank of America
Betfair
BMO Financial Group
Black Duck Software
Black Knight Financial Services
Box
Canadian Imperial Bank of Commerce
Capital One
City National Bank
Cisco
Citigroup
Citizen's Bank
Comerica Bank
Cryptography Research, a division of Rambus
Dahua
Depository Trust & Clearing Corporation
Ellucian
Experian
F-Secure
Fannie Mae
Fidelity
Freddie Mac
General Electric
Genetec
Global Payments
Highmark Health Solutions
Horizon Healthcare Services, Inc.
HSBC
Independent Health
iPipeline
Johnson & Johnson
JPMorgan Chase & Co.
Lenovo
LGE
McKesson
Medtronic
Morningstar
Navient
NCR
NetApp
NewsCorp
NVIDIA
NXP Semiconductors N.V.
PayPal
Principal Financial Group
Qualcomm
Royal Bank of Canada
Scientific Games
Sony Mobile
Splunk
Synopsys SIG
Target
TD Ameritrade
The Advisory Board
The Home Depot
The Vanguard Group
Trainline
Trane
U.S. Bank
Veritas
Verizon
Wells Fargo
Zendesk
Zephyr Health

Our thanks also to the more than 90 individuals who helped gather the data for BSIMM. In particular, we thank Mike Doyle, Nabil Hannan, Jason Hills, Brenton Kohler, Iman Louis, Nick Murison, Alistair Nash, Kevin Nassery, Denis Sheridan, and Mike Ware. In addition, 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. Resources for data analysis were provided by Oracle. BSIMM1–BSIMM3 were authored by Gary McGraw, Ph.D., Brian Chess, Ph.D., and Sammy Migues. BSIMM4–BSIMM8 were authored by Gary McGraw, Ph.D., Sammy Migues, and Jacob West.

BSIMM9 Table of Contents

BSIMM9 List of Tables

BSIMM9 List of Figures

PROLOGUE

What's New in BSIMM?

Dr. Gary McGraw, VP SECURITY TECHNOLOGY, SYNOPSYS

The BSIMM project is a de facto standard for assessing (and then improving) software security initiatives, and BSIMM9 is the culmination of a decade of objective, observation-based work in the field. Some of our more interesting findings include the following:

Cloud Transformation: Three new activities have been added to the BSIMM model that clearly show that software security in the cloud is becoming mainstream. Furthermore, activities observed among independent software vendors, Internet of Things companies, and cloud firms (three of our most distinct verticals) have begun to converge, suggesting that common cloud architectures require similar software security approaches.

Retail: A new vertical emerged in the BSIMM data pool. Software security initiatives are maturing relatively quickly as new models focused on e-commerce become critical to sustaining a healthy business.

Population Growth: The BSIMM now includes data from 120 firms; the number of developers it covers grew by 43 percent, and the number of software security practitioners it measures grew by 65 percent.

BSIMM9 incorporates the largest set of data collected about software security anywhere. By measuring your firm with the BSIMM measuring stick, you can directly compare and contrast your security approach to some of the best firms in the world.

It Takes a Community to Raise the Bar

Sammy Migues, PRINCIPAL SCIENTIST, SYNOPSYS

The industry as a whole and our community in particular have made great strides in the field of software security. Over the past 25 years, there have been fantastic improvements in design, language features, compilers, frameworks, protocols, and encryption. Tools help us write better code on the fly and find hundreds of security defects with relative ease. Organizational and process approaches create streamlined teams that can more easily respond to change. Every day, I meet people truly dedicated to software security improvement.

And yet, it seems as though the process of creating secure software and keeping it secure isn't significantly easier today.

The technology is often hard to use, introduces friction in automated processes, requires headcount to achieve the desired effectiveness, and improves much more slowly than software evolves. From a consumer perspective, the vendor marketplace is fragmented; you have to know exactly what you want so that you can get the best parts from multiple vendors and assemble them yourself. Tool output is often useful only to those who already know the answer, not to the people who don't understand security issues.

Moreover, processes are in constant flux. There is debt in nearly every organization, with approaches built for waterfall ecosystems being molded into agile forms or DevOps shapes. Increasingly, these processes are being stretched vanishingly thin into CI/CD toolchains. CI/CD requires automation in a way that waterfall never did. DevOps is a cultural change that can't be solved with any amount of incremental procedural alteration—in fact, improvements from agile processes often result in demand for even more output, and many business processes related to software value streams frequently have to be rebuilt from scratch.

In the midst of this turmoil, developers are expected to increase their output, application security architects are expected to create secure designs on the first try (and have them remain secure for years at a time), and testers are alternately expected to create huge test suites by knowing everything about the code and to just babysit the tools. More people are seeing their primary responsibilities automated away into technology that has none of the discretion that comes from years of experience.

Meanwhile, the bad guys are getting better, too. And did anyone notice that budgets are often tighter than we'd prefer?

This is all just a normal Tuesday in our industry. Things change. We react and improve. We're really good at it, and we continue to attract people to the profession. We've had 167 companies in our BSIMM community, each with a real, functioning software security initiative (SSI) addressing risk in software portfolios that often include hundreds or thousands of applications being worked on by thousands of developers. These SSIs deal with all this change—both from internal and external drivers—by creating their own technical solutions from a patchwork of open source and commercial tools. They integrate their security functions more tightly with development and operations to form new groups with shared objectives. They respond to changing definitions of security, privacy, and partner expectations with tighter integration into software requirements. Most importantly, these SSIs are real people who go to work every day and interact with thousands of other people to make a real difference in the software security world. They do a great job responding to everything going around them, and the BSIMM numbers show it.

A Decade of Software Security

Jacob West, VICE PRESIDENT OF CLOUD OPERATIONS, ORACLE

A decade ago, when the BSIMM was just the glimmer of an idea in the eyes of its creators, I was busy building technology designed to help organizations practice software security more effectively. Most firms that were aware of software security were plagued with insecurity about the level and type of investment they should make. Were they prioritizing their investments in the right way? Were they using enough technology? Too much technology? Many early leaders staked their own professional success on making the right choices, so they wondered what the future would hold both for the field and for their own careers.

Software security as a field was still in its fledgling stages at that time, and the only way to gauge the maturity of any given initiative was to hold it up against those in other organizations. Some firms published prescriptive guidance, like the Microsoft Security Development Lifecycle, but such efforts were too tightly coupled with their organization's way of developing software to serve as useful yardsticks for others. Small pockets of software security trailblazers learned from and shared with one another through ad hoc communities, but this approach created insular groups that were difficult to join and nearly impossible to scale.

Practitioners knew that technology investments such as static analysis could require years to implement at enterprise scale and were leery of making investments that might fail to provide adequate return. One question came up again and again: What would the state of the art in tooling for software security look like in a decade? At the time, I enjoyed pontificating that many of the specialized capabilities being developed would be subsumed by generic technologies, like compilers, integrated development environments, and even new languages.

> "Those who cannot remember the past are condemned to repeat it."
> —George Santayana

Fast forward ten years and I have both good news and bad news. Despite my youthful optimism, most technology designed to help with software security remains highly specialized. This means that organizations still face tough questions around how and when to deploy tools as part of their software security initiatives. However, armed with the BSIMM, they no longer suffer the uncertainty of answering those questions in a near vacuum.

The BSIMM was born out of necessity. The industry needed a yardstick for software security, not only to measure an organization's current state, but, more importantly, to govern investment over time. As the BSIMM enters its second decade, I firmly believe that its greatest value is as a historical document. Activities that were once rare and indicative of great maturity are now table stakes. Others have fallen by the wayside. And, perhaps most excitingly, readers can track new activities as they are introduced and adopted.

Through the raw data collected from 120 firms, BSIMM9 readers can not only understand how their organizations compare to their peers today, they can also understand how investment in software security has evolved over time. This historical understanding can in turn help their organizations predict where the state of the art will be in the future and then gauge their investments accordingly. As George Santayana put it, "Those who cannot remember the past are condemned to repeat it," and the BSIMM is the best way to understand where software security has been as well as where it's going.

PART ONE

The Building Security in Maturity Model (BSIMM, pronounced “bee simm”) is a study of software security initiatives. By quantifying the practices of many different organizations, we can describe the common ground shared by many as well as the variations that make each unique. Our aim is to help the wider software security community plan, carry out, and measure initiatives of their own. The BSIMM is not a “how to” guide, nor is it a one-size-fits-all prescription. Instead, the BSIMM is a reflection of the current state of software security.

We begin with a brief description of the function and importance of a software security initiative. We then explain our model and the method we use for quantifying the state of an initiative. Since the BSIMM study began in 2008, we have studied 167 firms, which comprise 389 distinct measurements (some firms use the BSIMM to measure each of their business units and some have been measured more than once). To ensure the continued relevance of the data we report, we excluded from BSIMM9 measurements older than 42 months. The current data set comprises 320 distinct measurements collected from 120 firms. Thanks to repeat measurements, not only do we report on current practices but also on the ways in which some initiatives have evolved over a period of years.

Later in this document, we give a detailed explanation of the key roles in a software security initiative, the 116 activities that now comprise our model, and a summary of the raw data we have collected. We have reviewed the description of each activity for BSIMM9.

Our work with the BSIMM shows that measuring a firm’s software security initiative is both possible and extremely useful. Organizations use their BSIMM measurements to plan, structure, and execute the evolution of a software security initiative. Over time, firms participating in the BSIMM show measurable improvement in their software security initiatives.

Over time, firms participating in the BSIMM show measurable improvement in their software security initiatives.

Introduction

History

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 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. We use the term software security initiative (SSI) to refer to all the activities undertaken for the purpose of building secure software.

BSIMM quantifies the activities carried out by real software security initiatives.

BSIMM9

The purpose of the BSIMM is to quantify the activities carried out by real software security initiatives. Because these initiatives use different methodologies and different terminology, the BSIMM requires a framework that allows us to describe all the initiatives 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 vertical markets, or create different work products.

We classify our work as a maturity model because improving software security almost always means changing the way an organization works, which doesn't happen overnight. We understand that not all organizations need to achieve the same security goals, but we believe all organizations can benefit from using the same measuring stick.

BSIMM9 is the ninth major version of the model. It includes updated activity descriptions, data from 120 firms in multiple vertical markets, and a longitudinal study.

Audience

The BSIMM is meant for use by anyone responsible for creating and executing an SSI. We have observed that successful SSIs are typically run by a senior executive who reports to the highest levels in an organization. These executives lead an internal group that we call the software security group (SSG), which is charged with directly executing or facilitating the activities described in the BSIMM. The BSIMM is written with the SSG and SSG leadership in mind.

We expect readers to be familiar with the software security literature. You can become familiar with many concepts by reading *Software Security: Building Security In*. The BSIMM does not attempt to explain software security basics, describe its history, or provide references to the ever-expanding literature. Succeeding with the BSIMM without becoming familiar with the literature is unlikely.

Method

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

- We relied on our own knowledge of software security practices to create the SSF. (We present the framework on page 13.)
- 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 show 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.

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 backward compatibility, we make all changes by adding new activity labels to the model, even when an activity has simply changed levels. We make changes by considering outliers both in the model itself and in the levels we assigned to various activities in the 12 practices we describe later. We use the results of an intralevel standard deviation analysis to determine which outlier activities to move between levels, focusing on changes that minimize standard deviation in the average number of observed activities at each level.

We use an in-person interview technique to conduct BSIMM assessments, done with a total of 167 firms so far. In 35 cases, we assessed the SSG and one or more business units as part of creating the corporate SSI view. In some of those cases, we used one aggregated scorecard, whereas in others, we used multiple scorecards for the SSG and each business unit. However, each firm is represented by only one set of data in the model published here. The following table shows changes in the data pool over time.

DATA POOL OVER TIME

ITERATION	NUMBER OF FIRMS AGED OUT	TOTAL NUMBER OF FIRMS
BSIMM-V	**5**	**67**
BSIMM6	**21**	**78**
BSIMM7	**13**	**95**
BSIMM8	**5**	**109**
BSIMM9	**11**	**120**

For BSIMM9, we added 22 firms and removed 11, resulting in a data pool of 120 firms. We used the resulting observation counts to refine the set of activities and their placement in the framework.

We have also conducted a second complete set of interviews with 42 of the current participating firms in order to study how their initiatives have changed over time. Twenty firms have undertaken three BSIMM assessments, seven have done four BSIMM assessments, and one has had five BSIMM assessments.

Simple observations, simply reported.

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 page 23). We also publish observations about subsets (such as industry verticals) when our sample size for the subset is large enough to guarantee anonymity.

As a descriptive model, the only goal of the BSIMM is to observe and report. We like to say that we wandered off into the jungle to see what we could see and discovered that "monkeys eat bananas in X of the Y jungles we visited." Note that the BSIMM does not report "you should only eat yellow bananas," "do not run while eating a banana," "thou shalt not steal thy neighbors' bananas," or any other value judgments. Simple observations, simply reported.

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 on this scale. Other work has either described the experience of a single organization or offered prescriptive guidance based purely on a combination of personal experience and opinion.

Participating Firms

The 120 participating organizations are drawn from eight well-represented verticals (with some overlap): financial services (50), independent software vendors (42), technology (22), healthcare (19), cloud (17), Internet of Things (16), insurance (10), and retail (10). Verticals with lower representation in the BSIMM population include telecommunications, security, and energy. See the Acknowledgments section on page 3 for a list of companies that graciously agreed to be identified.

On average, the 120 participating firms have practiced software security for 4.13 years at the time of the current assessment (ranging from less than a year to 19 years as of June 2018). All 120 firms agree that the success of their initiative hinges on their SSG, an internal group devoted to software security. SSG size on average is 13.3 people (smallest 1, largest 160, median 5.5), with an average satellite group of developers, architects, and people in the organization directly engaged in and promoting software security consisting of 52.4 people (smallest 0, largest 2,250, median 0). The average number of developers among our participants was 3,463 people (smallest 20, largest 45,000, median 900), yielding an average percentage of SSG to development of 1.33% (median 0.67%).

All told, the BSIMM describes the work of 1,600 SSG members working with a satellite of 6,291 people to secure the software developed by 415,598 developers as part of a combined portfolio of 135,881 applications.

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.

Domain: One of the four categories our framework is divided into: governance, intelligence, secure software development lifecycle (SSDL) touchpoints, and deployment. See the SSF section on page 13.

Practice: BSIMM activities are broken down into 12 categories or practices. Each domain in the software security framework (SSF) has three practices, and the activities in each practice are divided into an additional three levels. See the SSF section on page 13.

Satellite: A group of interested and engaged developers, architects, software managers, testers, and people in similar roles who have a natural affinity for software security and are 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. See the SSF section on page 13.

Software Security Group (SSG): The internal group charged with carrying out and facilitating software security. According to our observations, the first step of a software security initiative (SSI) is to form an SSG.

Software Security Initiative (SSI): An organization-wide program to instill, measure, manage, and evolve software security activities in a coordinated fashion. Also known in the literature as an Enterprise Software Security Program (see chapter 10 of *Software Security: Building Security In*).

BSIMM9 Structure

The BSIMM is organized as a set of 116 activities in a framework.

The Software Security Framework

The graphic below shows the SSF used to organize the 116 BSIMM activities. Twelve practices are organized into four domains.

The four domains:

Governance. Practices that help organize, manage, and measure a software security initiative. Staff development is also a central governance practice.

Intelligence. 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.

SSDL Touchpoints. Practices associated with analysis and assurance of particular software development artifacts and processes. All software security methodologies include these practices.

Deployment. Practices that interface with traditional network security and software maintenance organizations. Software configuration, maintenance, and other environment issues have direct impact on software security.

The 12 practices:

Governance

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

Intelligence

4. Attack Models (AM)
5. Security Features & Design (SFD)
6. Standards & Requirements (SR)

SSDL Touchpoints

7. Architecture Analysis (AA)
8. Code Review (CR)
9. Security Testing (ST)

Deployment

10. Penetration Testing (PT)
11. Software Environment (SE)
12. Configuration Management & Vulnerability Management (CMVM)


BSIMM9

The BSIMM9 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. It also includes the percentage of firms (out of 120) performing that activity in their own SSI. More complete descriptions of the activities, examples, and term definitions are available in Part Two of this document.

Governance

STRATEGY & METRICS (SM)

ACTIVITY DESCRIPTION	ACTIVITY	PARTICIPANT %
LEVEL 1		
Publish process (roles, responsibilities, plan), evolve as necessary.	SM1.1	59.2
Create evangelism role and perform internal marketing.	SM1.2	55.0
Educate executives.	SM1.3	55.8
Identify gate locations, gather necessary artifacts.	SM1.4	84.2
LEVEL 2		
Publish data about software security internally.	SM2.1	39.2
Enforce gates with measurements and track exceptions.	SM2.2	35.0
Create or grow a satellite.	SM2.3	36.7
Require security sign-off.	SM2.6	32.5
LEVEL 3		
Use an internal tracking application with portfolio view.	SM3.1	12.5
Run an external marketing program.	SM3.2	5.8
Identify metrics and use them to drive budgets.	SM3.3	15.0

COMPLIANCE & POLICY (CP)

ACTIVITY DESCRIPTION	ACTIVITY	PARTICIPANT %
LEVEL 1		
Unify regulatory pressures.	CP1.1	65.8
Identify PII obligations.	CP1.2	84.2
Create policy.	CP1.3	55.0

Table continued on next page >

Governance continued...

COMPLIANCE & POLICY (CP) ACTIVITY DESCRIPTION	ACTIVITY	PARTICIPANT %
LEVEL 2		
Identify PII data inventory.	CP2.1	32.5
Require security sign-off for compliance-related risk.	CP2.2	31.7
Implement and track controls for compliance.	CP2.3	35.8
Include software security SLAs in all vendor contracts.	CP2.4	35.0
Ensure executive awareness of compliance and privacy obligations.	CP2.5	39.2
LEVEL 3		
Create a regulator compliance story.	CP3.1	17.5
Impose policy on vendors.	CP3.2	10.0
Drive feedback from SSDL data back to policy.	CP3.3	4.2

TRAINING (T)

ACTIVITY DESCRIPTION	ACTIVITY	PARTICIPANT %
LEVEL 1		
Provide awareness training.	T1.1	66.7
Deliver role-specific advanced curriculum (tools, technology stacks, and bug parade).	T1.5	28.3
Create and use material specific to company history.	T1.6	21.7
Deliver on-demand individual training.	T1.7	39.2
LEVEL 2		
Enhance satellite through training and events.	T2.5	17.5
Include security resources in onboarding.	T2.6	19.2
LEVEL 3		
Reward progression through curriculum (certification or HR).	T3.1	3.3
Provide training for vendors or outsourced workers.	T3.2	6.7
Host external software security events.	T3.3	7.5
Require an annual refresher.	T3.4	7.5
Establish SSG office hours.	T3.5	4.2
Identify a satellite through training.	T3.6	2.5

Intelligence

ATTACK MODELS (AM)

ACTIVITY DESCRIPTION	ACTIVITY	PARTICIPANT %
LEVEL 1		
Create a data classification scheme and inventory.	AM1.2	62.5
Identify potential attackers.	AM1.3	31.7
Gather and use attack intelligence.	AM1.5	44.2
LEVEL 2		
Build attack patterns and abuse cases tied to potential attackers.	AM2.1	8.3
Create technology-specific attack patterns.	AM2.2	8.3
Build and maintain a top *N* possible attacks list.	AM2.5	13.3
Collect and publish attack stories.	AM2.6	11.7
Build an internal forum to discuss attacks.	AM2.7	9.2
LEVEL 3		
Have a science team that develops new attack methods.	AM3.1	3.3
Create and use automation to mimic attackers.	AM3.2	1.7

SECURITY FEATURES & DESIGN (SFD)

ACTIVITY DESCRIPTION	ACTIVITY	PARTICIPANT %
LEVEL 1		
Build and publish security features.	SFD1.1	79.2
Engage SSG with architecture.	SFD1.2	58.3
LEVEL 2		
Build secure-by-design middleware frameworks and common libraries.	SFD2.1	28.3
Create SSG capability to solve difficult design problems.	SFD2.2	38.3
LEVEL 3		
Form a review board or central committee to approve and maintain secure design patterns.	SFD3.1	7.5
Require use of approved security features and frameworks.	SFD3.2	7.5
Find and publish mature design patterns from the organization.	SFD3.3	1.7

Intelligence continued...

STANDARDS & REQUIREMENTS (SR)		
ACTIVITY DESCRIPTION	ACTIVITY	PARTICIPANT %
LEVEL 1		
Create security standards.	SR1.1	62.5
Create a security portal.	SR1.2	65.0
Translate compliance constraints to requirements.	SR1.3	63.3
LEVEL 2		
Create a standards review board.	SR2.2	31.7
Create standards for technology stacks.	SR2.3	19.2
Identify open source.	SR2.4	32.5
Create SLA boilerplate.	SR2.5	24.2
LEVEL 3		
Control open source risk.	SR3.1	14.2
Communicate standards to vendors.	SR3.2	8.3
Use secure coding standards.	SR3.3	8.3

SSDL Touchpoints

ARCHITECTURE ANALYSIS (AA)

ACTIVITY DESCRIPTION	ACTIVITY	PARTICIPANT %
LEVEL 1		
Perform security feature review.	AA1.1	84.2
Perform design review for high-risk applications.	AA1.2	27.5
Have SSG lead design review efforts.	AA1.3	22.5
Use a risk questionnaire to rank applications.	AA1.4	47.5
LEVEL 2		
Define and use AA process.	AA2.1	12.5
Standardize architectural descriptions (including data flow).	AA2.2	11.7
LEVEL 3		
Have software architects lead design review efforts.	AA3.1	3.3
Drive analysis results into standard architecture patterns.	AA3.2	1.7
Make the SSG available as an AA resource or mentor.	AA3.3	2.5

CODE REVIEW (CR)

ACTIVITY DESCRIPTION	ACTIVITY	PARTICIPANT %
LEVEL 1		
Have SSG perform ad hoc review.	CR1.2	68.3
Use automated tools along with manual review.	CR1.4	63.3
Make code review mandatory for all projects.	CR1.5	33.3
Use centralized reporting to close the knowledge loop and drive training.	CR1.6	36.7
LEVEL 2		
Assign tool mentors.	CR2.5	23.3
Use automated tools with tailored rules.	CR2.6	16.7
Use a top *N* bugs list (real data preferred).	CR2.7	20.8
LEVEL 3		
Build a factory.	CR3.2	3.3
Build a capability for eradicating specific bugs from the entire codebase.	CR3.3	0.8
Automate malicious code detection.	CR3.4	3.3
Enforce coding standards.	CR3.5	2.5

SSDL Touchpoints continued...

SECURITY TESTING (ST)		
ACTIVITY DESCRIPTION	ACTIVITY	PARTICIPANT %
LEVEL 1		
Ensure QA supports edge/boundary value condition testing.	ST1.1	83.3
Drive tests with security requirements and security features.	ST1.3	73.3
LEVEL 2		
Integrate black-box security tools into the QA process.	ST2.1	25.0
Share security results with QA.	ST2.4	11.7
Include security tests in QA automation.	ST2.5	10.0
Perform fuzz testing customized to application APIs.	ST2.6	10.8
LEVEL 3		
Drive tests with risk analysis results.	ST3.3	3.3
Leverage coverage analysis.	ST3.4	2.5
Begin to build and apply adversarial security tests (abuse cases).	ST3.5	2.5

Deployment

PENETRATION TESTING (PT)		
ACTIVITY DESCRIPTION	ACTIVITY	PARTICIPANT %
LEVEL 1		
Use external penetration testers to find problems.	PT1.1	87.5
Feed results to the defect management and mitigation system.	PT1.2	74.2
Use penetration testing tools internally.	PT1.3	61.7
LEVEL 2		
Provide penetration testers with all available information.	PT2.2	21.7
Schedule periodic penetration tests for application coverage.	PT2.3	17.5
LEVEL 3		
Use external penetration testers to perform deep-dive analysis.	PT3.1	8.3
Have the SSG customize penetration testing tools and scripts.	PT3.2	5.8

SOFTWARE ENVIRONMENT (SE)		
ACTIVITY DESCRIPTION	ACTIVITY	PARTICIPANT %
LEVEL 1		
Use application input monitoring.	SE1.1	48.3
Ensure host and network security basics are in place.	SE1.2	86.7
LEVEL 2		
Publish installation guides.	SE2.2	32.5
Use code signing.	SE2.4	25.8
LEVEL 3		
Use code protection.	SE3.2	14.2
Use application behavior monitoring and diagnostics.	SE3.3	3.3
Use application containers.	SE3.4	9.2
Use orchestration for containers and virtualized environments.	SE3.5	0.0
Enhance application inventory with operations bill of materials.	SE3.6	0.0
Ensure cloud security basics.	SE3.7	0.0

Deployment continued...

CONFIGURATION MANAGEMENT & VULNERABILITY MANAGEMENT (CMVM)		
ACTIVITY DESCRIPTION	ACTIVITY	PARTICIPANT %
LEVEL 1		
Create or interface with incident response.	CMVM1.1	84.2
Identify software defects found in operations monitoring and feed them back to development.	CMVM1.2	85.0
LEVEL 2		
Have emergency codebase response.	CMVM2.1	68.3
Track software bugs found in operations through the fix process.	CMVM2.2	72.5
Develop an operations inventory of applications.	CMVM2.3	47.5
LEVEL 3		
Fix all occurrences of software bugs found in operations.	CMVM3.1	4.2
Enhance the SSDL to prevent software bugs found in operations.	CMVM3.2	5.8
Simulate software crises.	CMVM3.3	7.5
Operate a bug bounty program.	CMVM3.4	10.8

Putting BSIMM9 to Use

The BSIMM describes 116 activities that any organization can put into practice. The activities are structured in terms of the SSF, which identifies 12 practices grouped into four domains.

What BSIMM9 Tells Us

The BSIMM data yield very interesting analytical results. The BSIMM9 scorecard on the following page shows the number of times each of the 116 activities briefly outlined in the BSIMM skeleton was observed in the BSIMM9 data. These are the highest-resolution BSIMM data that are published.

> "The BSIMM is a fundamental resource for those looking for solid foundations or improvement for their software security initiative. It is a consistent, systematic approach to classifying and understanding real data about actual activities of security-conscious organizations worldwide."
>
> –Iván Arce, CTO, Quarkslab

BSIMM9 SCORECARD

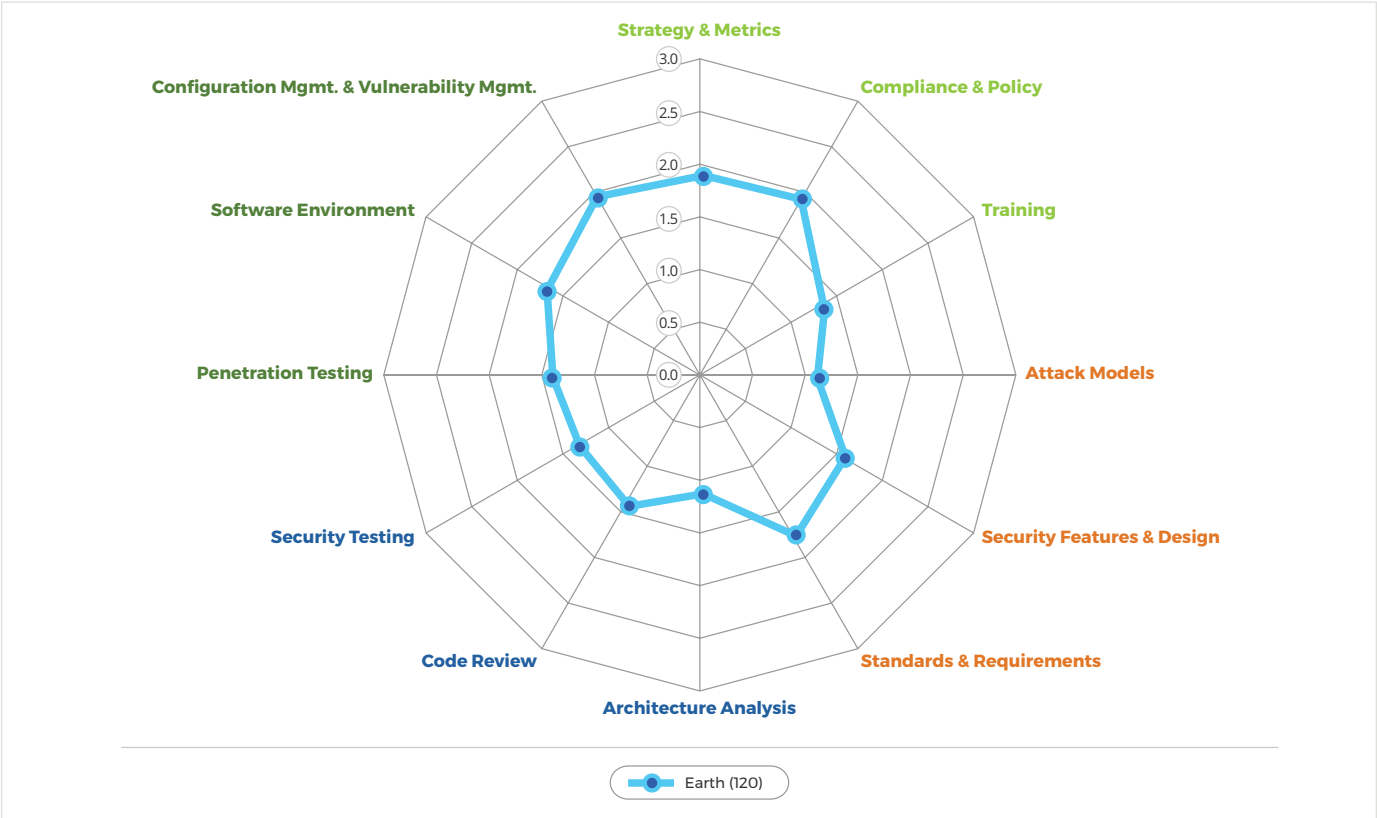
GOVERNANCE		INTELLIGENCE		SSDL TOUCHPOINTS		DEPLOYMENT	
ACTIVITY	BSIMM9 FIRMS (out of 120)	ACTIVITY	BSIMM9 FIRMS (out of 120)	ACTIVITY	BSIMM9 FIRMS (out of 120)	ACTIVITY	BSIMM9 FIRMS (out of 120)
Strategy & Metrics		**Attack Models**		**Architecture Analysis**		**Penetration Testing**	
[SM1.1]	71	[AM1.2]	75	[AA1.1]	101	[PT1.1]	105
[SM1.2]	66	[AM1.3]	38	[AA1.2]	33	[PT1.2]	89
[SM1.3]	67	[AM1.5]	53	[AA1.3]	27	[PT1.3]	74
[SM1.4]	101	[AM2.1]	10	[AA1.4]	57	[PT2.2]	26
[SM2.1]	47	[AM2.2]	10	[AA2.1]	15	[PT2.3]	21
[SM2.2]	42	[AM2.5]	16	[AA2.2]	14	[PT3.1]	10
[SM2.3]	44	[AM2.6]	14	[AA3.1]	4	[PT3.2]	7
[SM2.6]	39	[AM2.7]	11	[AA3.2]	2		
[SM3.1]	15	[AM3.1]	4	[AA3.3]	3		
[SM3.2]	7	[AM3.2]	2				
[SM3.3]	18						
Compliance & Policy		**Security Features & Design**		**Code Review**		**Software Environment**	
[CP1.1]	79	[SFD1.1]	95	[CR1.2]	82	[SE1.1]	58
[CP1.2]	101	[SFD1.2]	70	[CR1.4]	76	[SE1.2]	104
[CP1.3]	66	[SFD2.1]	34	[CR1.5]	40	[SE2.2]	39
[CP2.1]	39	[SFD2.2]	46	[CR1.6]	44	[SE2.4]	31
[CP2.2]	38	[SFD3.1]	9	[CR2.5]	28	[SE3.2]	17
[CP2.3]	43	[SFD3.2]	9	[CR2.6]	20	[SE3.3]	4
[CP2.4]	42	[SFD3.3]	2	[CR2.7]	25	[SE3.4]	11
[CP2.5]	47			[CR3.2]	4	[SE3.5]	0
[CP3.1]	21			[CR3.3]	1	[SE3.6]	0
[CP3.2]	12			[CR3.4]	4	[SE3.7]	0
[CP3.3]	5			[CR3.5]	3		
Training		**Standards & Requirements**		**Security Testing**		**Config. Mgmt. & Vuln. Mgmt.**	
[T1.1]	80	[SR1.1]	75	[ST1.1]	100	[CMVM1.1]	101
[T1.5]	34	[SR1.2]	78	[ST1.3]	88	[CMVM1.2]	102
[T1.6]	26	[SR1.3]	76	[ST2.1]	30	[CMVM2.1]	82
[T1.7]	47	[SR2.2]	38	[ST2.4]	14	[CMVM2.2]	87
[T2.5]	21	[SR2.3]	23	[ST2.5]	12	[CMVM2.3]	57
[T2.6]	23	[SR2.4]	39	[ST2.6]	13	[CMVM3.1]	5
[T3.1]	4	[SR2.5]	29	[ST3.3]	4	[CMVM3.2]	7
[T3.2]	8	[SR3.1]	17	[ST3.4]	3	[CMVM3.3]	9
[T3.3]	9	[SR3.2]	10	[ST3.5]	3	[CMVM3.4]	13
[T3.4]	9	[SR3.3]	10				
[T3.5]	5						
[T3.6]	3						

In the table on page 23, we also identified the most common activity in each practice (shown in yellow in the scorecard). These 12 activities were observed in at least 75 (62%) of the 120 firms we studied, and they appear in the table below. Although we can't directly conclude that these 12 activities are necessary for all SSIs, we can say with confidence that these activities are commonly found in highly successful initiatives. This suggests that if you are working on an initiative of your own, you should consider these 12 activities particularly carefully.

TWELVE CORE ACTIVITIES THAT "EVERYBODY" DOES	
ACTIVITY	DESCRIPTION
[SM1.4]	**Identify gate locations and gather necessary artifacts.**
[CP1.2]	**Identify PII obligations.**
[T1.1]	**Provide awareness training.**
[AM1.2]	**Create a data classification scheme and inventory.**
[SFD1.1]	**Build and publish security features.**
[SR1.2]	**Create a security portal.**
[AA1.1]	**Perform security feature review.**
[CR1.2]	**Have SSG perform ad hoc review.**
[ST1.1]	**Ensure QA supports 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.**
[CMVM1.2]	**Identify software bugs found in operations monitoring and feed them back to development.**

We created spider charts by noting the highest-level activity observed for each practice per BSIMM firm (a "high-water mark") and then averaging these values over a group of firms to produce 12 numbers (one for each practice). The resulting spider chart plots these values on 12 spokes corresponding to the 12 practices. Note that level 3 (the outside edge) is considered more mature than level 0 (the center point). Other, more sophisticated analyses are possible, of course.

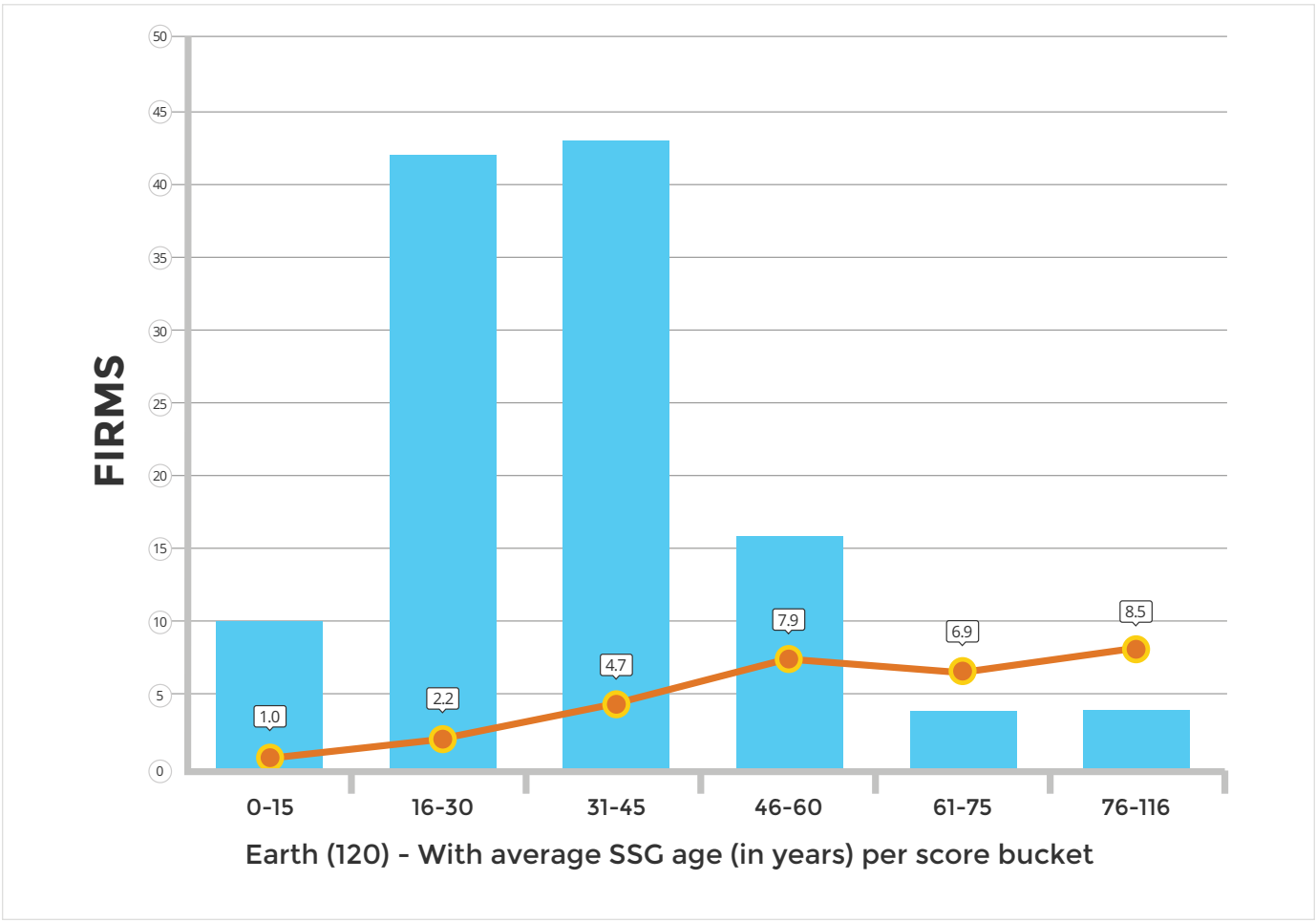
EARTH SPIDER CHART



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, 79].

The graph on the next page shows the distribution of scores among the population of 120 participating firms (which we call Earth). 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 orange line on the graph. In general, firms where more BSIMM activities have been observed have older SSIs.

BSIMM9 SCORE DISTRIBUTION



We are pleased that the BSIMM study continues to grow year after year. The data set we report on here is over 35 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.

Measuring Your Firm with BSIMM9

The most important use of the BSIMM is as a measuring stick to determine where your approach currently stands relative to other firms. You can simply note which activities you already have in place, find them in the skeleton, determine their levels, and then build your scorecard. A direct comparison of all 116 activities is perhaps the most obvious use of the BSIMM. This can be accomplished by building your scorecard and comparing it to the data above.

The scorecard you see on the next page depicts a fake firm that performs 37 BSIMM activities (noted as 1's in the FAKEFIRM columns), including seven activities that are the most common in their respective practices (purple boxes). Note the firm does not perform the most commonly observed activities in the other five practices (red boxes) and should take some time to determine whether these are necessary or useful to its overall software security initiative. The BSIMM9 FIRMS columns show the number of observations (currently out of 120) for each activity, allowing the firm to understand the general popularity of an activity among the 120 BSIMM9 firms.

BSIMM9 SCORECARD FOR: FAKEFIRM | OBSERVATIONS: 37

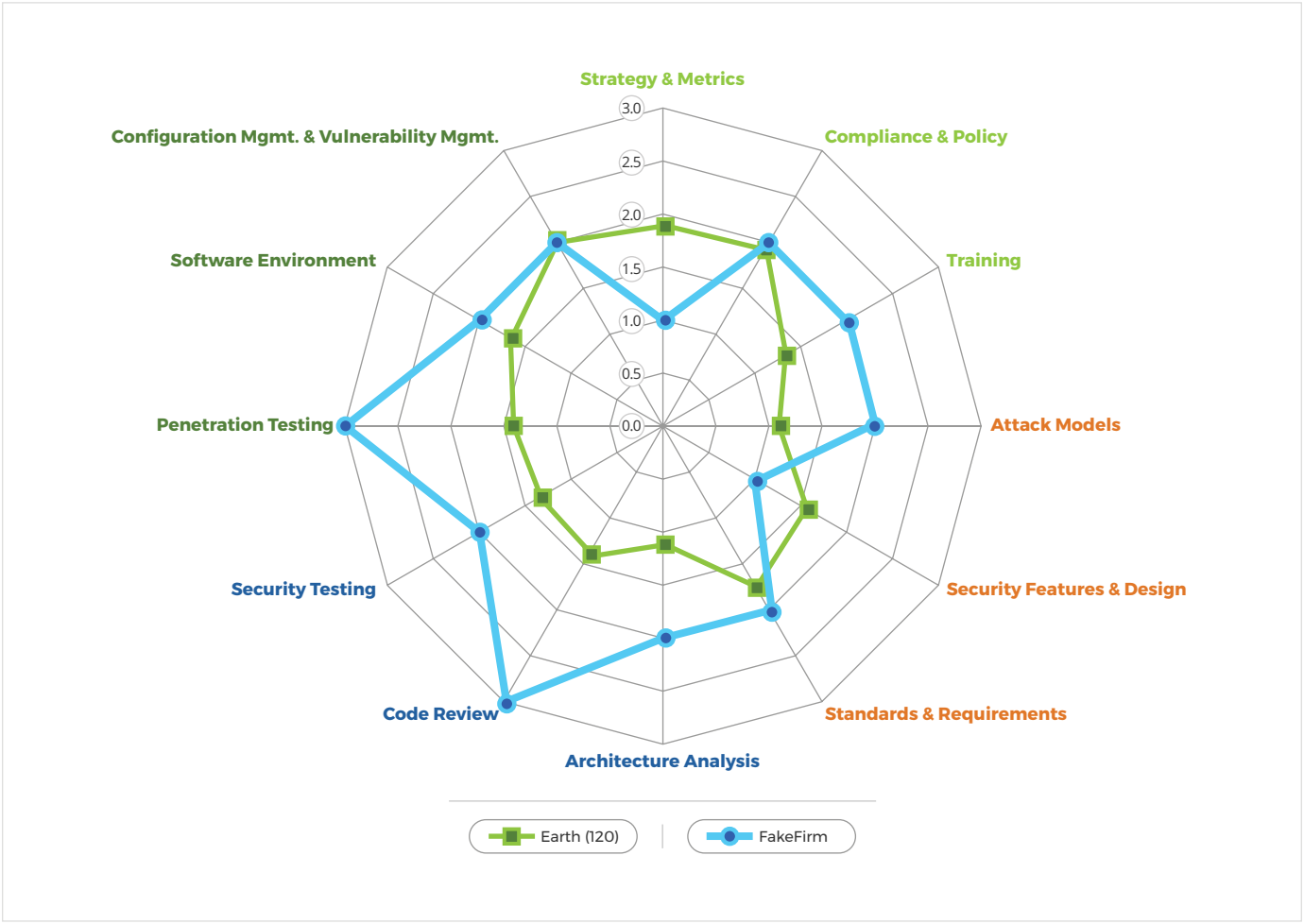
GOVERNANCE			INTELLIGENCE			SSDL TOUCHPOINTS			DEPLOYMENT		
ACTIVITY	BSIMM9 FIRMS (120)	FAKEFIRM	ACTIVITY	BSIMM9 FIRMS (120)	FAKEFIRM	ACTIVITY	BSIMM9 FIRMS (120)	FAKEFIRM	ACTIVITY	BSIMM9 FIRMS (120)	FAKEFIRM
Strategy & Metrics			**Attack Models**			**Architecture Analysis**			**Penetration Testing**		
[SM1.1]	71	1	[AM1.2]	75		[AA1.1]	101	1	[PT1.1]	105	1
[SM1.2]	66		[AM1.3]	38		[AA1.2]	33	1	[PT1.2]	89	1
[SM1.3]	67	1	[AM1.5]	53	1	[AA1.3]	27	1	[PT1.3]	74	
[SM1.4]	101	1	[AM2.1]	10		[AA1.4]	57		[PT2.2]	26	1
[SM2.1]	47		[AM2.2]	10	1	[AA2.1]	15		[PT2.3]	21	
[SM2.2]	42		[AM2.5]	16	1	[AA2.2]	14	1	[PT3.1]	10	1
[SM2.3]	44		[AM2.6]	14	1	[AA3.1]	4		[PT3.2]	7	
[SM2.6]	39		[AM2.7]	11		[AA3.2]	2				
[SM3.1]	15		[AM3.1]	4		[AA3.3]	3				
[SM3.2]	7		[AM3.2]	2							
[SM3.3]	18										
Compliance & Policy			**Sec. Features & Design**			**Code Review**			**Software Environment**		
[CP1.1]	79	1	[SFD1.1]	95		[CR1.2]	82	1	[SE1.1]	58	
[CP1.2]	101		[SFD1.2]	70	1	[CR1.4]	76	1	[SE1.2]	104	1
[CP1.3]	66	1	[SFD2.1]	34		[CR1.5]	40		[SE2.2]	39	1
[CP2.1]	39		[SFD2.2]	46		[CR1.6]	44	1	[SE2.4]	31	
[CP2.2]	38		[SFD3.1]	9		[CR2.5]	28		[SE3.2]	17	
[CP2.3]	43		[SFD3.2]	9		[CR2.6]	20		[SE3.3]	4	
[CP2.4]	42		[SFD3.3]	2		[CR2.7]	25		[SE3.4]	11	
[CP2.5]	47	1				[CR3.2]	4	1	[SE3.5]	0	
[CP3.1]	21					[CR3.3]	1		[SE3.6]	0	
[CP3.2]	12					[CR3.4]	4		[SE3.7]	0	
[CP3.3]	5					[CR3.5]	3				
Training			**Standards & Requirements**			**Security Testing**			**Config. Mgmt. & Vuln. Mgmt.**		
[T1.1]	80	1	[SR1.1]	75	1	[ST1.1]	100	1	[CMVM1.1]	101	1
[T1.5]	34		[SR1.2]	78		[ST1.3]	88	1	[CMVM1.2]	102	
[T1.6]	26	1	[SR1.3]	76	1	[ST2.1]	30	1	[CMVM2.1]	82	1
[T1.7]	47		[SR2.2]	38	1	[ST2.4]	14		[CMVM2.2]	87	
[T2.5]	21		[SR2.3]	23		[ST2.5]	12		[CMVM2.3]	57	
[T2.6]	23	1	[SR2.4]	39	1	[ST2.6]	13		[CMVM3.1]	5	
[T3.1]	4		[SR2.5]	29		[ST3.3]	4		[CMVM3.2]	7	
[T3.2]	8		[SR3.1]	17		[ST3.4]	3		[CMVM3.3]	9	
[T3.3]	9		[SR3.2]	10		[ST3.5]	3		[CMVM3.4]	13	
[T3.4]	9		[SR3.3]	10							
[T3.5]	5										
[T3.6]	3										

LEGEND:		
	ACTIVITY	116 BSIMM9 activities, shown in 4 domains and 12 practices
	BSIMM9 FIRMS	count of firms (out of 120) observed performing each activity
		the 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 BSIMM9 average

Once you have determined where you stand with activities, you can devise a plan to enhance practices with other activities included in the BSIMM. 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 activities that make sense for your organization and ignore those that don't, but revisit those choices periodically.

In our own work using the BSIMM to assess initiatives, we found that creating a spider chart yielding a high-water mark approach (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.

EARTH VS. FAKE FIRM SPIDER CHART



One meaningful comparison is to chart your own high-water mark against the averages we have published to see how your initiative stacks up. Above, we have plotted data from the fake firm against the BSIMM Earth graph. The breakdown of activities into levels for each practice is meant only as a guide. The levels provide 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 higher level in the same practice. That said, the levels we have identified hold water under statistical scrutiny. Level 1 activities (straightforward and simple) are those that are most commonly observed, Level 2 (more difficult and requiring more coordination) are slightly less so, and Level 3 (rocket science) are rarely observed.

By identifying activities from each practice that could work 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 a real budget, mandate, and ownership behind them. Although all initiatives look different and are tailored to fit a particular organization, all initiatives share common core activities, as we describe on page 24.

BSIMM9 Analysis

The BSIMM has produced a wealth of real-world data about software security.

BSIMM Over Time

This is the ninth major release of the BSIMM, and the chart below shows how it 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 data set.) BSIMM9 describes the work of 7,891 SSG and satellite people working directly in software security, impacting the security efforts of 415,598 developers.

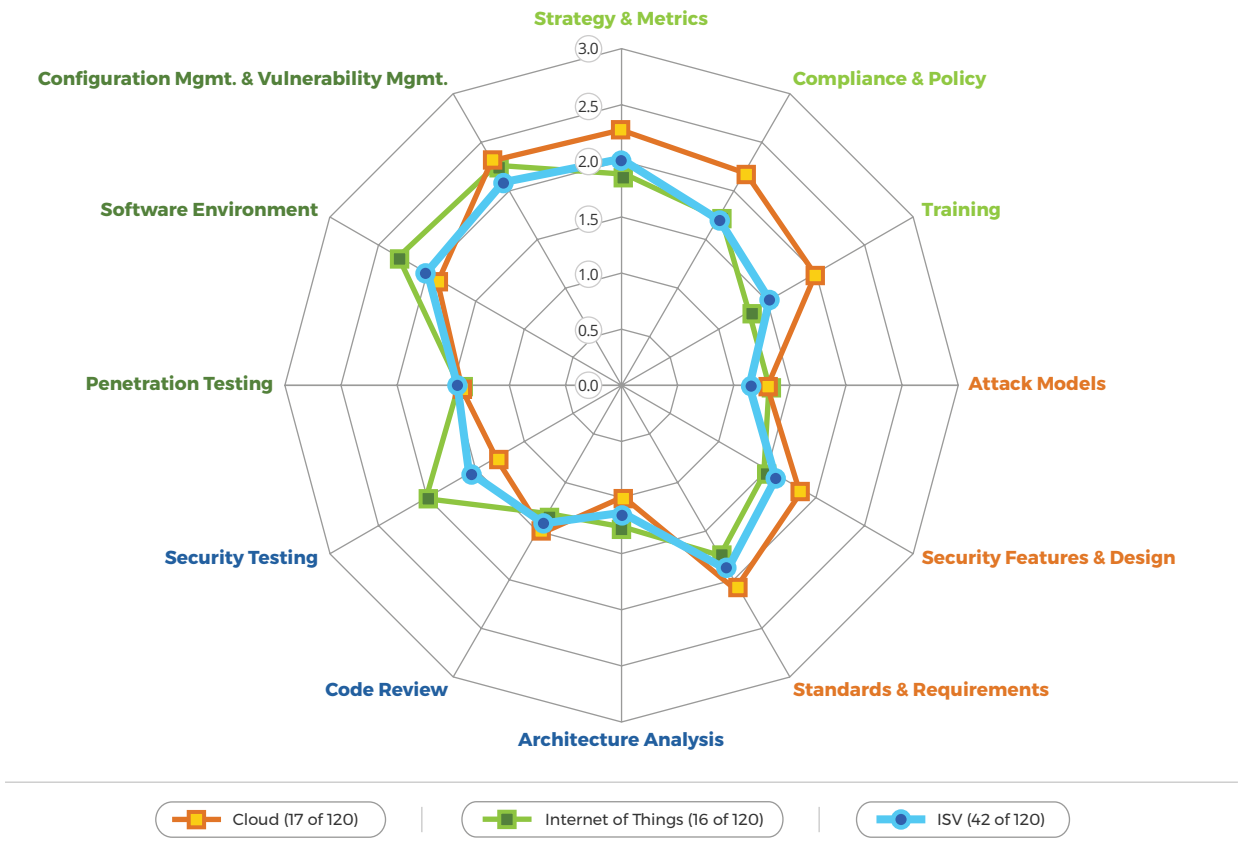
BSIMM NUMBERS OVER TIME

	BSIMM9	BSIMM8	BSIMM7	BSIMM6	BSIMM-V	BSIMM4	BSIMM3	BSIMM2	BSIMM1
FIRMS	120	109	95	78	67	51	42	30	9
MEASUREMENTS	320	256	237	202	161	95	81	49	9
2ND MEASURES	42	36	30	26	21	13	11	0	0
3RD MEASURES	20	16	15	10	4	1	0	0	0
4TH MEASURES	7	5	2	2	0	0	0	0	0
SSG MEMBERS	1,600	1,268	1,111	1,084	976	978	786	635	370
SATELLITE MEMBERS	6,291	3,501	3,595	2,111	1954	2039	1750	1150	710
DEVELOPERS	415,598	290,582	272,782	287,006	272,358	218,286	185,316	141,175	67,950
APPLICATIONS	135,881	94,802	87,244	69,750	69,039	58,739	41,157	28,243	3,970
AVG. SSG AGE (YEARS)	4.13	3.88	3.94	3.98	4.28	4.13	4.32	4.49	5.32
SSG AVG. OF AVGS	1.33 / 100	1.60 /100	1.61 / 100	1.51 / 100	1.40 / 100	1.95 / 100	1.99 / 100	1.02 / 100	1.13 / 100
FINANCIAL SERVICES	50	47	42	33	26	19	17	12	4
ISVs	42	38	30	27	25	19	15	7	4
TECH	22	16	14	17	14	13	10	7	2
HEALTHCARE	19	17	15	10					
INTERNET OF THINGS	16	12	12	13					
CLOUD	17	16	15						
INSURANCE	10	11	10						
RETAIL	10								

BSIMM and Industry Verticals

The spider charts we introduced earlier are also useful for comparing groups of firms from particular industry verticals. The following graphs show data charted together from verticals well represented in the BSIMM: financial services (50), independent software vendors (42), healthcare (19), cloud (17), Internet of Things (16), insurance (10), and retail (10).

CLOUD VS. INTERNET OF THINGS VS. ISV SPIDER CHART

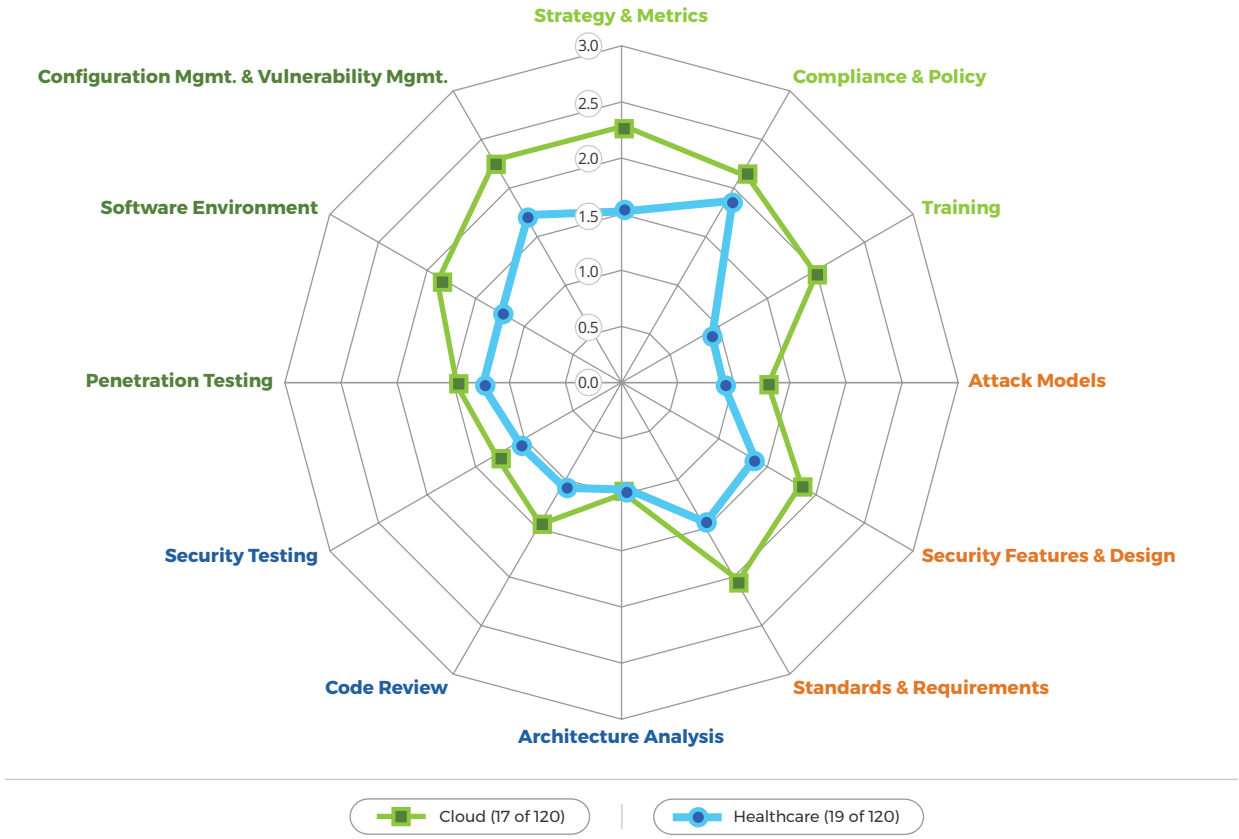


Cloud, Internet of Things, and independent software vendors (ISVs) are three of the most mature verticals in the BSIMM. On average, cloud firms are noticeably more mature in the Governance practices—Strategy & Metrics, Compliance & Policy, and Training—compared to the ISVs and Internet of Things firms. By the same measure, Internet of Things firms show greater maturity in the Security Testing and Software Environment practices. Despite these obvious differences, there is a great deal of overlap. We hypothesize that technology stacks and architectures between these three verticals are converging.

INSURANCE VS. HEALTHCARE VS. FINANCIAL SPIDER CHART

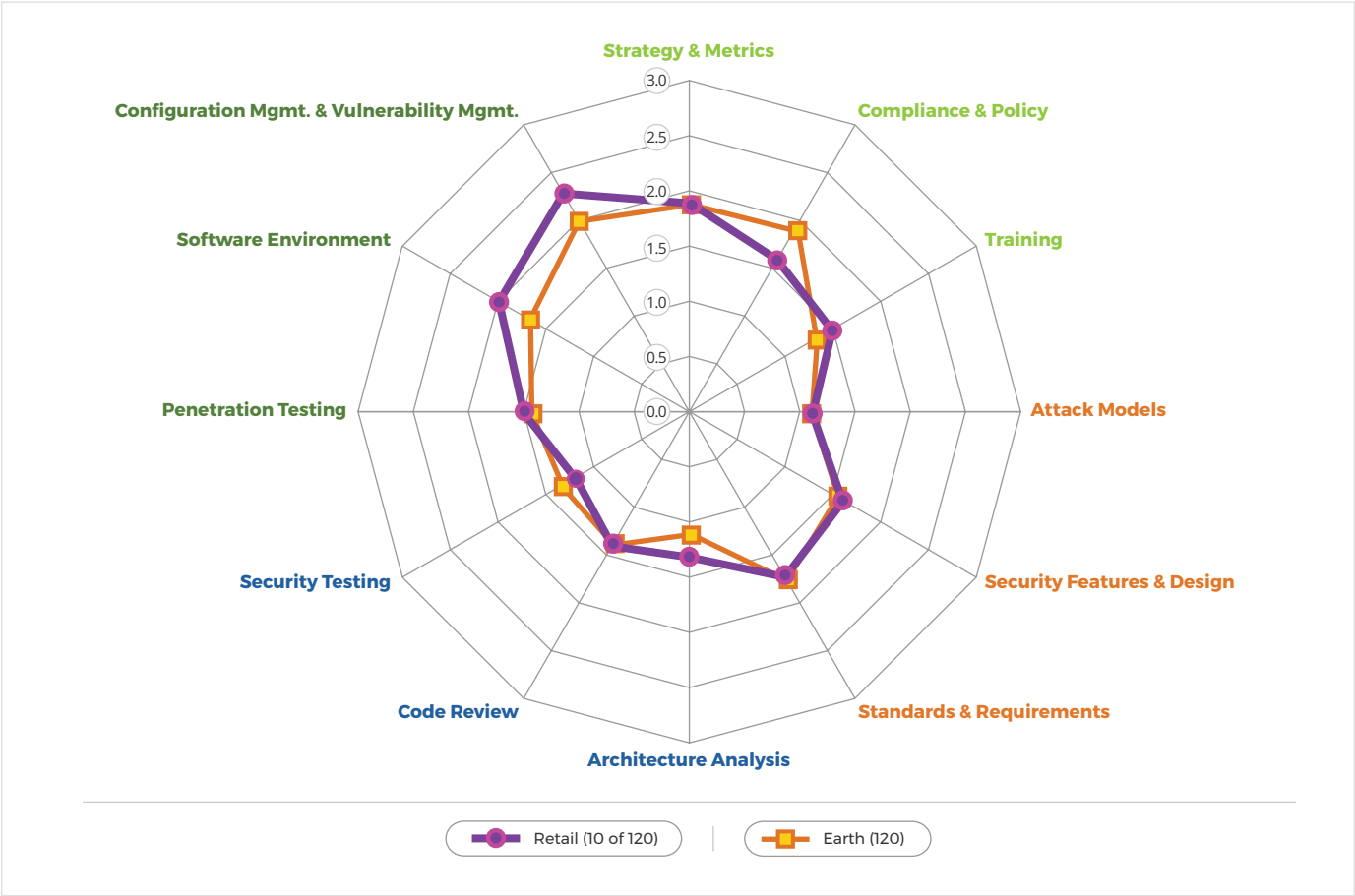
Three verticals in the BSIMM operate in highly regulated industries: insurance, healthcare, and financial services. In our experience, large financial services firms reacted to regulatory changes and started their SSIs much earlier than insurance and healthcare firms. Even as the number of financial services firms doubled over the past five years with a large influx into the BSIMM of newly-started initiatives, the financial services SSG average age at assessment time remains 5.4 years, versus 3.1 years for insurance and 2.5 years for healthcare. Time spent maturing their collective SSIs shows up clearly in the side-by-side comparison. Although the insurance vertical includes some mature outliers, the data for these three regulated verticals show insurance generally lags behind in software security. We see a starker contrast in healthcare, with virtually no outliers. The overall maturity of the healthcare vertical remains low.

CLOUD VS. HEALTHCARE SPIDER CHART



In the BSIMM population, we can find large gaps between the maturity of verticals. Consider the spider diagram that directly compares the cloud and healthcare verticals. In this case, the delta between technology firms that deliver cloud services and healthcare firms that are generally just getting started with software security is rather obvious. Fortunately for verticals that find themselves behind this curve, verticals such as cloud provide a good roadmap to faster maturity.

RETAIL VS. EARTH SPIDER CHART



For the first time, the BSIMM presents data on the retail vertical. This group, with an average SSG age of 3.2 years and average SSG size of nearly eight full-time people, seems to track closely to the overall data pool. The most obvious differences are in the Architecture Analysis, Software Environment, and Configuration Management & Vulnerability Management practices, where retail participants are somewhat ahead of the average for Earth.

In the tables on the following pages, you can see the BSIMM scorecards for the seven verticals compared side by side. In the Activity columns, we have highlighted in yellow the most common activity in each practice as observed in the entire BSIMM data pool (120 firms).

VERTICAL COMPARISON SCORECARD

GOVERNANCE								
Activity	Financial (of 50)	ISV (of 42)	Tech (of 22)	Healthcare (of 19)	IoT (of 16)	Insurance (of 10)	Cloud (of 17)	Retail (of 10)
[SM1.1]	36	25	12	10	9	5	11	5
[SM1.2]	25	28	17	8	13	4	12	5
[SM1.3]	30	26	14	8	10	5	12	3
[SM1.4]	47	34	18	13	11	9	13	10
[SM2.1]	24	19	9	4	6	4	10	5
[SM2.2]	25	12	8	4	5	3	7	3
[SM2.3]	16	17	11	8	8	5	5	5
[SM2.6]	23	9	6	4	4	2	6	3
[SM3.1]	7	6	3	2	2	1	4	0
[SM3.2]	1	5	3	1	2	1	3	1
[SM3.3]	12	4	2	2	2	1	2	0
[CP1.1]	37	24	13	16	12	6	11	6
[CP1.2]	45	29	18	19	14	8	16	10
[CP1.3]	37	18	7	10	5	5	10	5
[CP2.1]	18	13	6	8	6	2	9	3
[CP2.2]	22	9	7	7	4	2	5	1
[CP2.3]	19	14	10	6	6	3	7	4
[CP2.4]	21	14	7	7	5	3	8	2
[CP2.5]	18	19	9	11	8	3	10	2
[CP3.1]	15	7	1	2	1	2	4	0
[CP3.2]	6	3	2	2	1	2	4	0
[CP3.3]	3	1	1	0	0	0	1	0
[T1.1]	36	31	15	10	12	6	15	8
[T1.5]	16	14	7	3	5	3	8	3
[T1.6]	10	14	8	3	7	0	6	2
[T1.7]	26	15	8	5	6	6	8	5
[T2.5]	8	8	4	2	3	2	3	3
[T2.6]	15	5	3	2	3	2	5	2
[T3.1]	0	3	2	0	1	0	3	0
[T3.2]	4	3	3	2	3	2	3	1
[T3.3]	1	5	3	1	1	1	2	0
[T3.4]	5	3	1	2	1	1	3	0
[T3.5]	1	2	0	0	0	0	2	2
[T3.6]	0	2	2	0	2	0	2	0

INTELLIGENCE								
Activity	Financial (of 50)	ISV (of 42)	Tech (of 22)	Healthcare (of 19)	IoT (of 16)	Insurance (of 10)	Cloud (of 17)	Retail (of 10)
[AM1.2]	41	20	11	10	11	6	12	6
[AM1.3]	18	11	7	6	6	3	3	4
[AM1.5]	26	16	11	8	8	4	7	4
[AM2.1]	2	4	3	2	2	1	2	1
[AM2.2]	2	5	5	0	3	0	4	0
[AM2.5]	6	5	6	2	3	1	2	1
[AM2.6]	4	5	3	3	3	1	5	0
[AM2.7]	2	6	6	1	4	0	4	0
[AM3.1]	1	2	2	0	1	0	1	1
[AM3.2]	0	1	2	0	1	0	0	0
[SFD1.1]	42	32	16	14	12	9	16	8
[SFD1.2]	29	28	17	12	13	6	12	6
[SFD2.1]	14	16	8	4	6	2	8	2
[SFD2.2]	18	18	10	6	8	4	8	4
[SFD3.1]	6	2	1	0	1	0	2	1
[SFD3.2]	4	4	0	1	0	2	5	1
[SFD3.3]	0	0	1	1	1	0	0	1
[SR1.1]	41	23	12	11	10	5	12	5
[SR1.2]	36	27	13	14	10	5	14	6
[SR1.3]	40	24	15	9	10	7	9	8
[SR2.2]	23	11	6	4	3	4	6	4
[SR2.3]	13	6	4	4	3	2	5	3
[SR2.4]	15	19	10	6	8	3	10	1
[SR2.5]	14	9	6	6	5	3	4	2
[SR3.1]	7	8	6	1	4	2	5	1
[SR3.2]	3	4	4	2	3	3	2	0
[SR3.3]	4	3	4	2	4	2	1	0

SSDL TOUCHPOINTS								
Activity	Financial (of 50)	ISV (of 42)	Tech (of 22)	Healthcare (of 19)	IoT (of 16)	Insurance (of 10)	Cloud (of 17)	Retail (of 10)
[AA1.1]	43	36	19	15	14	7	15	10
[AA1.2]	10	15	11	3	8	2	6	2
[AA1.3]	6	13	10	4	7	2	6	1
[AA1.4]	34	11	8	10	7	5	5	6
[AA2.1]	4	8	6	2	3	2	1	0
[AA2.2]	4	5	6	2	5	1	1	1
[AA3.1]	2	1	2	0	1	1	1	0
[AA3.2]	0	1	1	0	1	0	1	1
[AA3.3]	1	1	2	0	1	0	1	0
[CR1.2]	34	29	16	12	9	7	12	6
[CR1.4]	35	27	14	9	11	5	13	7
[CR1.5]	16	20	11	2	6	3	7	2
[CR1.6]	21	18	7	4	6	2	9	3
[CR2.5]	15	11	4	3	4	2	6	4
[CR2.6]	15	3	3	0	2	1	2	1
[CR2.7]	13	8	2	2	1	3	4	1
[CR3.2]	1	0	0	2	0	2	0	1
[CR3.3]	0	0	1	0	1	0	0	0
[CR3.4]	4	0	0	0	0	0	0	0
[CR3.5]	2	0	1	0	1	0	0	0
[ST1.1]	44	34	21	12	14	10	12	9
[ST1.3]	41	34	20	9	13	7	11	7
[ST2.1]	11	15	10	4	8	4	3	3
[ST2.4]	7	4	5	1	2	1	0	1
[ST2.5]	4	5	6	1	4	0	4	1
[ST2.6]	0	11	11	1	8	0	3	0
[ST3.3]	1	3	1	0	1	0	2	0
[ST3.4]	0	1	3	1	3	0	0	0
[ST3.5]	1	2	1	0	1	0	1	0

DEPLOYMENT								
Activity	Financial (of 50)	ISV (of 42)	Tech (of 22)	Healthcare (of 19)	IoT (of 16)	Insurance (of 10)	Cloud (of 17)	Retail (of 10)
[PT1.1]	45	36	18	16	13	10	14	10
[PT1.2]	43	34	13	8	10	6	15	7
[PT1.3]	31	25	13	12	9	6	9	8
[PT2.2]	7	10	8	3	5	2	4	2
[PT2.3]	13	9	2	1	1	1	3	2
[PT3.1]	1	5	7	2	4	1	2	1
[PT3.2]	4	2	2	0	1	0	1	0
[SE1.1]	30	13	4	13	5	5	11	7
[SE1.2]	47	35	17	17	13	9	16	9
[SE2.2]	14	19	15	2	11	2	7	2
[SE2.4]	8	17	16	1	11	1	5	1
[SE3.2]	5	8	8	2	6	2	2	3
[SE3.3]	0	3	2	0	2	0	1	0
[SE3.4]	3	7	2	0	1	0	4	3
[SE3.5]	0	0	0	0	0	0	0	0
[SE3.6]	0	0	0	0	0	0	0	0
[SE3.7]	0	0	0	0	0	0	0	0
[CMVM1.1]	44	38	19	13	15	6	16	8
[CMVM1.2]	42	38	19	16	15	7	15	9
[CMVM2.1]	40	31	13	11	10	5	13	7
[CMVM2.2]	37	33	18	11	14	5	15	10
[CMVM2.3]	28	22	9	9	8	5	9	2
[CMVM3.1]	0	3	3	0	3	0	2	0
[CMVM3.2]	1	2	4	2	3	1	2	0
[CMVM3.3]	5	1	3	2	1	1	0	2
[CMVM3.4]	4	6	2	1	1	2	6	2

BSIMM as a Longitudinal Study

Forty-two of the 120 firms in BSIMM9 have been measured at least twice. On average, the time between their first and second measurements was 27.1 months. Although individual activities among the 12 practices come and go (as shown in the longitudinal scorecard on the next page), in general, remeasurement over time shows a clear trend of increased maturity among the 42 firms remeasured thus far. The raw score went up in 34 of the 42 firms, and remained the same in three firms. Across all 42 firms, the observation count increased by an average of 10.3 (39.8%). SSIs mature over time.

LONGITUDINAL SCORECARD

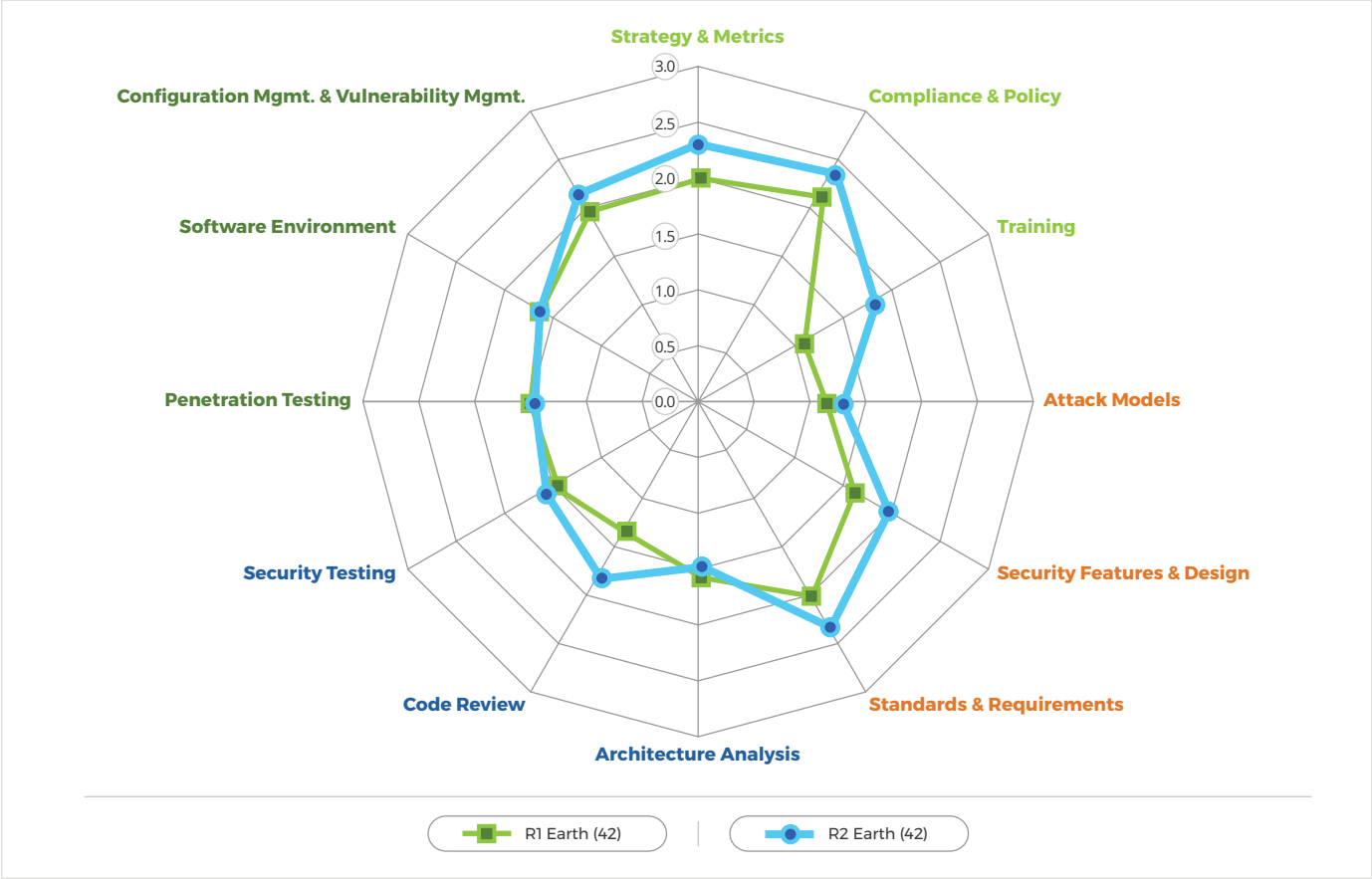
GOVERNANCE			INTELLIGENCE			SSDL TOUCHPOINTS			DEPLOYMENT		
Activity	BSIMM Round 1 (of 42)	BSIMM Round 2 (of 42)	Activity	BSIMM Round 1 (of 42)	BSIMM Round 2 (of 42)	Activity	BSIMM Round 1 (of 42)	BSIMM Round 2 (of 42)	Activity	BSIMM Round 1 (of 42)	BSIMM Round 2 (of 42)
[SM1.1]	17	34	[AM1.2]	28	35	[AA1.1]	36	40	[PT1.1]	37	40
[SM1.2]	20	27	[AM1.3]	11	18	[AA1.2]	13	16	[PT1.2]	21	36
[SM1.3]	20	26	[AM1.5]	18	24	[AA1.3]	10	15	[PT1.3]	21	27
[SM1.4]	34	39	[AM2.1]	4	6	[AA1.4]	24	28	[PT2.2]	9	7
[SM2.1]	15	25	[AM2.2]	4	4	[AA2.1]	4	8	[PT2.3]	15	13
[SM2.2]	12	18	[AM2.5]	6	8	[AA2.2]	2	3	[PT3.1]	4	2
[SM2.3]	15	21	[AM2.6]	7	4	[AA3.1]	5	4	[PT3.2]	1	2
[SM2.6]	18	22	[AM2.7]	4	6	[AA3.2]	0	1			
[SM3.1]	7	12	[AM3.1]	0	1	[AA3.3]	7	4			
[SM3.2]	1	2	[AM3.2]	0	1						
[SM3.3]	10	12									
[CP1.1]	27	31	[SFD1.1]	33	37	[CR1.2]	23	31	[SE1.1]	18	22
[CP1.2]	35	38	[SFD1.2]	28	32	[CR1.4]	25	35	[SE1.2]	36	38
[CP1.3]	22	33	[SFD2.1]	10	17	[CR1.5]	13	18	[SE2.2]	14	15
[CP2.1]	15	19	[SFD2.2]	11	21	[CR1.6]	15	26	[SE2.4]	7	12
[CP2.2]	15	16	[SFD3.1]	2	7	[CR2.5]	9	17	[SE3.2]	1	4
[CP2.3]	15	17	[SFD3.2]	6	10	[CR2.6]	4	13	[SE3.3]	7	2
[CP2.4]	13	21	[SFD3.3]	3	2	[CR2.7]	10	13	[SE3.4]	0	2
[CP2.5]	20	25				[CR3.2]	2	3	[SE3.5]	0	0
[CP3.1]	8	14				[CR3.3]	1	3	[SE3.6]	0	0
[CP3.2]	9	10				[CR3.4]	0	0	[SE3.7]	0	0
[CP3.3]	1	3				[CR3.5]	4	3			
[T1.1]	27	36	[SR1.1]	28	34	[ST1.1]	30	35	[CMVM1.1]	34	38
[T1.5]	7	18	[SR1.2]	25	34	[ST1.3]	31	34	[CMVM1.2]	39	38
[T1.6]	8	8	[SR1.3]	25	34	[ST2.1]	15	16	[CMVM2.1]	36	38
[T1.7]	14	26	[SR2.2]	13	22	[ST2.4]	3	7	[CMVM2.2]	27	35
[T2.5]	5	11	[SR2.3]	0	15	[ST2.5]	2	5	[CMVM2.3]	17	28
[T2.6]	7	8	[SR2.4]	10	17	[ST2.6]	5	4	[CMVM3.1]	2	0
[T3.1]	1	4	[SR2.5]	9	17	[ST3.3]	1	1	[CMVM3.2]	1	3
[T3.2]	1	5	[SR3.1]	5	7	[ST3.4]	0	0	[CMVM3.3]	2	1
[T3.3]	0	2	[SR3.2]	6	8	[ST3.5]	3	3	[CMVM3.4]	0	5
[T3.4]	1	7	[SR3.3]	13	10						
[T3.5]	1	6									
[T3.6]	3	4									

There are two ways of thinking about the change represented by the longitudinal scorecard (showing 42 BSIMM9 firms moving from their first to second assessment). We see the biggest changes in the following activities: *[SM1.1 Publish process (roles, responsibilities, plan), evolve as necessary]*, with 18 new observations; *[PT1.2 Feed results to the defect management and mitigation system]*, with 16 new observations; *[T1.7 Deliver on-demand individual training]* and *[CMVM2.3 Develop an operations inventory of applications]*, each with 15 new

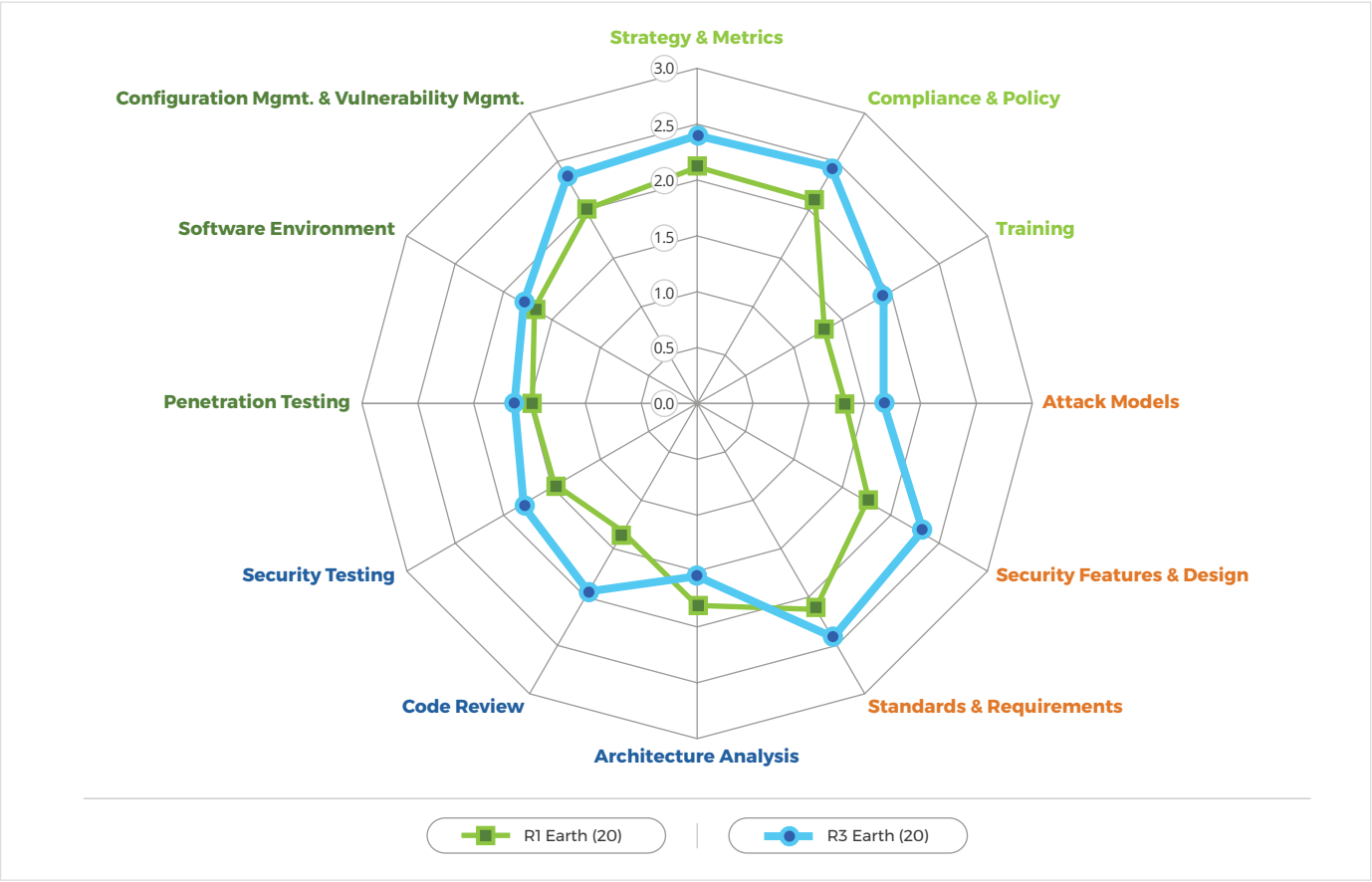
observations; *[SM2.1 Publish data about software security internally]*, with 14 new observations; and *[T1.5 Deliver role-specific advanced curriculum (tools, technology stacks, and bug parade)]*, *[SM1.2 Create evangelism role and perform internal marketing]*, *[SFD2.2 Create SSG capability to solve difficult design problems]*, and *[SR1.2 Create a security portal]*, each with 13 new observations. The change cannot always be seen directly on this scorecard. For example, *[CP2.3 Implement and track controls for compliance]* was a new activity for nine firms, but the scorecard clearly shows that the total observations increased by only two. What happened? Simply put, there was churn. Although the activity was newly observed in nine firms, it was no longer observed in seven firms or went away due to data aging out, giving a total change of two (as shown in the scorecard).

In a different example, the activity *[T1.6 Create and use material specific to company history]* was both newly observed in four firms and no longer observed in four firms. Therefore, the total observation count remains unchanged on the scorecard. The same type of zero-sum churn also occurred in *[AM2.2 Create technology-specific attack patterns]*.

ROUND 1 EARTH VS. ROUND 2 EARTH



ROUND 1 EARTH VS. ROUND 3 EARTH



Firms tend to mature between measurements, as seen in the two spider charts on pages 36 and 37. Forty-two firms have been measured twice, and 20 firms have been measured three times.

Emerging Trends in the BSIMM Data

As the BSIMM community grew, we added a greater number of firms with newer software security initiatives, and we began to track new verticals that have less software security experience. Thus, we expected to see a direct impact on the data. Specifically, adding firms with less experience decreased average overall maturity 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. One reason for this change—a potential reversal of the decline in overall maturity—appears to be the mix of firms embarking on their first BSIMM assessment. The average SSG age for new firms entering BSIMM6 was 2.9 years; it was 3.37 years for BSIMM7 and 2.83 years for BSIMM8, but increased to 4.57 years for BSIMM9. Another reason appears to be an increase in firms continuing to use BSIMM assessments to guide their initiatives. BSIMM7 included 11 firms that received their second or higher assessment. That figure increased to 12 firms for BSIMM8 and 16 firms for BSIMM9.

We also see this potential reversal 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, the average financial services score increased to 36.8. Note that five of the 11 firms dropped from BSIMM9 due to data age were

in the financial services group. A similar effect was evident in personnel where, with the exception of some outliers, we observed an overall decrease in SSG size on first measurement, but the number increased from 6.1 for BSIMM7 and 8.8 for BSIMM8 to 11.6 for BSIMM9. 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 (66 of 120 firms had no satellite at the time of their current assessment, and two-thirds of the firms added for BSIMM9 had no satellite at assessment time). Previous BSIMM reports indicated that the existence of a satellite is directly related to maturity, thus the fact that fewer firms had a satellite accounted for the number of immature firms in the BSIMM8 population. For the 54 BSIMM9 firms with a satellite at assessment time, the average size was 116.5.

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. 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. This kind of change can be seen in activities spanning all 12 practices. Instead of focusing on a robust, multiactivity approach to a given practice, many firms have a tendency to pick one figurehead activity on which to focus their next round of investment.

Firms in the BSIMM community for multiple years have, with one or two exceptions, always increased in maturity over time. We expect the majority of newer firms entering the BSIMM population to do the same.

BSIMM Community

The 120 firms participating in the BSIMM make up the BSIMM community. A private online community platform with nearly 600 members provides software security personnel a forum to discuss solutions with others who face the same issues, refine strategy with someone who has already addressed an issue, seek out mentors from those further along a career path, and band together to solve hard problems. Community members also receive exclusive access to topical webinars and other curated content.

The BSIMM community also hosts annual private conferences where representatives from each firm gather in an off-the-record forum to discuss software security initiatives. To date, 15 BSIMM community conferences have been held, eight in the United States and seven in Europe. During the conferences, representatives from BSIMM firms give the presentations.

The *BSIMM website* includes a credentialed BSIMM community section where information from conferences, working groups, and mailing list-initiated studies are posted.

PART TWO

This part of the document provides detail behind the people and activities involved in a software security initiative and measured by the BSIMM. We begin by describing the software security group (SSG) and other key roles. We then provide descriptions for each of the 116 activities that comprise BSIMM9. Throughout, we annotate our discussion with relevant observation data from the 120 participating firms.

Roles in a Software Security Initiative

Determining who is supposed to carry out the activities described in the BSIMM is an important part of making any SSI work.

Executive Leadership

Of primary interest is identifying and empowering a senior executive to manage operations, garner resources, and provide political cover for an SSI. Grassroots approaches to software security sparked and led solely by developers and their direct managers have a poor track record in the real world. Likewise, initiatives spearheaded by resources from an existing network security group often run into serious trouble when it comes time to interface with development groups. By identifying a senior executive and putting him or her in charge of software security directly, you address two management 101 concerns: accountability and empowerment. You also create a place in the organization where software security can take root and begin to thrive.

The individuals in charge of the SSIs we studied have a variety of titles. Examples include Chief Data & Security Privacy Officer, CISO, CSO, Director Enterprise Security Architecture, Director Global Security & Compliance, Director Product Security, Executive Director Product Operations, Head Application Security Architecture & Engineering, Head Application Security Programs, Manager InfoSec Engineering, Manager Product Security, Managing VP of Security Engineering, Threat & Vulnerability Management Lead, VP Application Security & Technology Analysis, VP Cybersecurity, VP InfoSec, and Web Security Manager. We observed a fairly wide spread in exactly where the SSG is situated in the firms we studied. In particular, 63 of the 120 participating firms have SSGs that are run by a CISO or report to a CISO as their nearest senior executive. Thirteen of the firms report through a CTO as their closest senior executive, while 10 report to a CIO, seven to a CSO, four to a COO, two to a CRO, and one to a CAO. Twenty of the SSGs report through some type of technology or product organization.

> Developer-led grassroots approaches to software security have a poor track record in the real world.

Software Security Group (SSG)

The second most important role in an SSI after the senior executive is the SSG itself. Every single one of the 120 initiatives we describe in BSIMM9 has an SSG. In fact, successfully carrying out BSIMM activities without an SSG is very unlikely (and has never been observed in the field to date), 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 impossible to find. If you must create software security types from scratch, start with developers and teach them about security. Starting with network security people 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 come in a variety of shapes and sizes, but all good SSGs appear to include both people with deep coding experience and people with architectural chops. As you will see below, software security can't only be about finding specific bugs, such as the *OWASP Top Ten*. Code review is an important best practice, and to perform code review, you must actually understand code (not to mention the huge piles of security bugs). However, the best code reviewers sometimes make poor software architects, and asking them to perform an architecture risk analysis will only result in blank stares. Make sure that you cover architectural capabilities in your SSG as well as you cover code. Finally, SSGs are often asked to mentor, train, and work directly with hundreds of developers. Communication skills, teaching capability, and practical knowledge are must-haves for at least a portion of the SSG staff. For more about this issue, see our Search Security article based on SSG structure data gathered at the 2014 BSIMM Community Conference: *How to Build a Team for Software Security Management*.

Although no two of the 120 firms we examined had exactly the same SSG structure (suggesting that there is no one set way to structure an SSG), we did observe some commonalities that are worth mentioning. At the highest level of organization, SSGs come in five major flavors: 1) organized to provide software security services, 2) organized around setting policy, 3) mirroring business unit organizations, 4) organized with a hybrid policy and services approach, and 5) structured around managing a distributed network of others doing software security work. Some SSGs are highly distributed across a firm and others are centralized. If we look across all of the SSGs in our study, though, there are several common subgroups: 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; and vendor control groups.

In the statistics reported above, we noted an average ratio of SSG to development of 1.33% across the entire group of 120 organizations that we studied, meaning we found one SSG member for every 75 developers when we averaged the ratios for each participating firm. The SSG with the largest ratio was 10%, and the smallest was 0.01%. As a reminder, SSG size on average among the 120 firms was 13.3 people (smallest 1, largest 160, median 5.5).

Satellite

In addition to the SSG, many SSIs have identified a number of individuals (often developers, testers, and architects) who share a basic interest in software security but are not directly employed in the SSG. When people like this carry out software security activities, we call this group a satellite.

Satellites are sometimes widely distributed, with one or two members in each product group, and sometimes they are more focused, getting together regularly to compare notes, learn new technologies, and expand the understanding of an organization's software security. 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, 27 of the 30 firms with the highest BSIMM scores have a satellite, with an average satellite size of nearly 182 people. Outside the top 30, 27 of the remaining 90 firms have a satellite (30%). Of the 30 firms with the lowest BSIMM scores, only three have a satellite, and the bottom 10 have no satellite at all.

Sixty-four percent of firms that have been assessed more than once have a satellite, while 65% of firms on their first assessment do not. 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. Among our population of 120 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

Our survey participants have engaged everyone involved in the software development lifecycle (SDLC) as a means of addressing software security.

Twenty-seven of the 30 firms with the highest BSIMM scores have a satellite.

- **Builders**, including developers, architects, and their managers, must practice security engineering, ensuring that the systems that they build are defensible and not riddled with holes. The SSG will interact directly with builders when they carry out the activities described in the BSIMM. Generally speaking, as an organization matures, the SSG attempts to empower builders so that they can carry out most BSIMM activities themselves, with the SSG helping in special cases and providing oversight. 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 lifecycle.
- **Testers** concerned with routine testing and verification should do what they can to keep an eye out for security problems. Some BSIMM activities in the Security Testing practice can be carried out directly by QA.
- **Operations** people must continue to design, defend, and maintain reasonable environments. As you will see in the Deployment domain of the software security framework (SSF), software security doesn't end when software is "shipped." This includes cloud software and DevOps shops.
- **Administrators** must understand the distributed nature of modern systems 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 line of 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. Any sizeable business today depends on software to work. Software security is a business necessity.
- **Vendors**, including those who supply COTS, custom software, and software-as-a-service, are increasingly subjected to SLAs and reviews (such as *vBSIMM*) that help ensure products are the result of a secure SDLC.

BSIMM9 Activities

Take a look at the BSIMM skeleton in Part One if you need help understanding how the activities associated with each of the 12 BSIMM practices are organized in this section. For each activity, we show the activity label, followed by the number of firms in BSIMM9 performing that activity.

GOVERNANCE: Strategy & Metrics (SM)

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

SM LEVEL 1

[SM1.1: 71] Publish process (roles, responsibilities, plan), evolve as necessary.

The process for addressing software security is broadcast to all stakeholders so that everyone knows the plan. Goals, roles, responsibilities, and activities are explicitly defined. Most organizations pick and choose from a published methodology, such as the Microsoft SDL or the Synopsys Touchpoints, and then tailor the methodology to their needs. An SSDL process must be adapted to the specifics of the development process it governs (e.g., waterfall, agile, CI/CD, DevOps, etc.) because it will evolve with both the organization and the security landscape. A process must be published to count. In many cases, the methodology is controlled by the SSG and published only internally. The SSDL does not need to be publicly promoted outside of the firm to have the desired impact.

[SM1.2: 66] Create evangelism role and perform internal marketing.

In order to build support for software security throughout the organization, someone in the SSG must play an evangelism role. This internal marketing function helps keep executives and all other stakeholders current on the magnitude of the software security problem and the elements of its solution. An agile coach familiar with security, for example, could help teams adopt better software security practices as they transform to an agile methodology. Evangelists typically give talks for internal groups including executives, extend invitations to outside speakers, author white papers for internal consumption, or create a collection of papers, books, and other resources on an internal website and promote its use. A canonical example of such an evangelist was Michael Howard's role at Microsoft just after the Gates memo.

[SM1.3: 67] Educate executives.

Executives are periodically shown the consequences of inadequate software security and the negative business impact that poor security can have. They're also shown what other organizations are doing to attain software security, including dealing with the risks of adopting "flavor of the day" engineering methodologies with no oversight. By understanding both the problem and its proper resolution, executives can support the SSI as a risk management necessity. In its most dangerous form, such 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., actually showing working exploits and their business impact). In some cases, presentation to the Board can help garner resources for an ongoing SSI. Bringing in an outside guru is often helpful when seeking to bolster executive attention. Tying education to specific development areas, such as mobile or cloud services, or particular methodologies, such as CI/CD and DevOps, can help convince leadership to accept SSG recommendations where they might otherwise be ignored in favor of faster release dates or other priorities.

[SM1.4: 101] Identify gate locations, gather necessary artifacts.

The software security process includes release gates/checkpoints/milestones at one or more points in an SDLC or, more likely, multiple SDLCs. The first two steps toward establishing security-specific release gates are to identify gate locations that are compatible with existing development practices and to begin gathering the input necessary for making a go/no-go decision. Importantly, the gates are not enforced at this stage. For example, the SSG can collect security testing results for each project prior to release but stop short of passing judgment on what constitutes sufficient testing or acceptable test results. Shorter release cycles, as are seen in organizations practicing CI/CD, often require creative approaches to collecting the right evidence and rely heavily on lightweight, super-fast automation. The idea of identifying gates first and enforcing them later on is extremely helpful in moving development toward software security without major pain. Socialize the gates and then turn them on once most projects already know how to succeed. This gradual approach serves to motivate good behavior without requiring it.

SM LEVEL 2

[SM2.1: 47] Publish data about software security internally.

The SSG publishes data internally about the state of software security within the organization to facilitate improvement. This information might come in the form of a dashboard with metrics for executives and software development management. Sometimes, these published data are not shared with everyone in the firm but with the relevant executives only. In such cases, publishing the information to executives who then drive change in the organization is necessary. In other cases, open book management and data published to all stakeholders helps everyone know what's going on, with the philosophy that sunlight is the best disinfectant. If the organization's culture promotes internal competition between groups, this information adds a security dimension to the game. The time compression associated with CI/CD calls for measurements that can be taken quickly and accurately, focusing less on historical trends (e.g., bugs per release) and more on speed (e.g., time to fix).

[SM2.2: 42] Enforce gates with measurements and track exceptions.

SDLC security gates are enforced for every software project: to pass a gate, a project must either meet an established measure or obtain a waiver. Even recalcitrant project teams must now play along. The SSG tracks exceptions. A gate could require a project to undergo code review and remediate any critical findings before release. In some cases, gates are directly associated with controls required by regulations, contractual agreements, and other business obligations, and exceptions are tracked as required by statutory or regulatory drivers. In other cases, gate measures yield key performance indicators that are used to govern the process. A revolving door or a rubber stamp exception process does not count. If some projects are automatically passed, that defeats the purpose of enforcing gates. Even seemingly innocuous development projects, such as a new mobile client for an existing back-end or an application ported to a cloud environment from an internal data center, must successfully pass the prescribed security gates in order to progress. Similarly, APIs, frameworks, libraries, COTS, microservices, container configurations, and so on are all software that must traverse the security gates.

A revolving door or a rubber stamp exception process does not count.

[SM2.3: 44] Create or grow a satellite.

A satellite begins as a collection of people scattered across the organization who show an above-average level of security interest or skill. Identifying this group is a step toward creating a social network that speeds the adoption of security into software development. One way to begin is to track the people who stand out during introductory training courses (see *[T3.6 Identify satellite through training]*). Another way is to ask for volunteers. In a more top-down approach, initial satellite membership is assigned to ensure complete coverage of all development/product groups. Ongoing membership should be based on actual performance. A strong satellite is a good sign of a mature SSI. In new or fast-moving technology areas such as mobile development, or in development paradigms such as DevOps, satellite members can help combine software security skills with domain knowledge that might be underrepresented in the SSG. Agile coaches make particularly useful satellite members.

> The key is to tie technical results to business objectives.

[SM2.6: 39] Require security sign-off.

The organization has an initiative-wide process for accepting security risk and documenting accountability. A risk acceptor signs off on the state of all software prior to release. For example, the sign-off policy might require the head of the business unit to sign off on critical vulnerabilities that have not been mitigated or SSDL steps that have been skipped. The policy must apply to outsourced projects, such as a boutique mobile application, and to projects that will be deployed in external environments, such as the cloud. Informal or uninformed risk acceptance alone does not count as security sign-off because the act of accepting risk is more effective when it is formalized (e.g., with a signature, form submission, or something similar) and captured for future reference. Similarly, simply stating that certain projects never need a sign-off does not achieve the desired results.

SM LEVEL 3

[SM3.1: 15] Use an internal tracking application with portfolio view.

The SSG uses a centralized tracking application to chart the progress of every piece of software in its purview, regardless of development methodology. The application records the security activities scheduled, in progress, and completed, incorporating results from activities such as architecture analysis, code review, and security testing even when they happen in a tight loop. The SSG uses the tracking application to generate portfolio reports for many of the metrics it uses. A combined inventory and risk posture view is fundamental. In many cases, these data are published 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: 7] Run an external marketing program.

The SSG helps the firm market the SSI outside to build external support. Software security grows beyond being a risk reduction exercise and instead becomes a competitive advantage or market differentiator. The SSG might write papers or books about its SSDL or have a public blog. It might also participate in external conferences or trade shows. In some cases, a complete SSDL methodology can be published and promoted externally. Mobile and cloud security projects can make great software security case studies. Sharing details externally and inviting critique can bring new perspectives into the firm.

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

The SSG and its management choose the metrics that define and measure SSI progress. These metrics will drive the initiative's budget and resource allocations, so simple counts and statistics won't suffice. Metrics also allow the SSG to explain its goals and progress in quantitative terms. One such metric could be security defect density, a reduction in which could be used to show a decreasing cost of remediation over time. Recall that, in agile methodologies, metrics are best collected early and often in a lightweight manner. The key here is to tie technical 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.

> **Software security should grow beyond a risk reduction exercise to become a competitive advantage.**

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 service-level agreements (SLAs) to help control COTS software risk, setting organizational software security policy, and auditing against that policy.

CP LEVEL 1

[CP1.1: 79] Unify regulatory pressures.

If the business or its customers are subject to regulatory or compliance drivers such as GDPR, FFIEC, GLBA, OCC, PCI DSS, SOX, HIPAA, or others, the SSG acts as a focal point for understanding the constraints such drivers impose on software. In some cases, the SSG creates a unified approach that removes redundancy and conflicts from overlapping compliance requirements. A formal approach will map applicable portions of regulations to control statements explaining 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 single set of software security guidance ensures that compliance work is completed as efficiently as possible. Some firms move on to guide exposure by becoming directly involved in standards groups exploring new technologies in order to influence the regulatory environment.

[CP1.2: 101] Identify PII obligations.

The way software handles personally identifiable information (PII) could be explicitly regulated, but even if it isn't, privacy is a hot topic. The SSG plays a key role in identifying and describing PII obligations stemming from regulation and customer expectations. It uses this information to promote best practices related to privacy. For example, if the organization processes credit card transactions, the SSG will identify 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) does not relax PII obligations. Also note that firms creating software products that process PII (but that don't necessarily handle PII directly) can get credit by providing privacy

controls and guidance for their customers. The proliferation of Internet of Things (IoT) and mobile devices adds yet another dimension to PII protection.

[CP1.3: 66] 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. The policy 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 don't need to relearn customer security requirements on their own. The SSG policy documents are sometimes focused around major compliance topics such as the handling of PII or the use of cryptography. In some cases, policy documents relate directly to the SSDL and its use in the firm. Because they're new, codifying decisions about IoT, cloud, and mobile architectures can add some much needed pizzazz to the policy discussion. Similarly, it may be necessary to explain what can and can't be automated into CI/CD and continuous deployment pipelines. Architecture standards and coding guidelines are not examples of software security policy. On the other hand, policy that prescribes and mandates the use of coding guidelines and architecture standards for certain categories of applications does count. Policy is what is permitted and denied at the initiative level. If it's not mandatory, it's not policy.

> If it's not mandatory, it's not policy.

CP LEVEL 2

[CP2.1: 39] Identify PII data inventory.

The organization identifies the kinds of PII processed or stored by each of its systems and their data repositories, including mobile and cloud environments. A PII inventory can be approached in two ways: starting with each individual application by noting its PII use or starting with particular types of PII and the applications that touch them. In either case, an inventory of data repositories is required. Note that when applications are distributed across multiple deployment environments, PII inventory control can get tricky. Do not ignore it. Likewise, do not ignore the constantly evolving definitions of PII. When combined with the organization's PII obligations, this inventory guides privacy planning. For example, the SSG can now create a list of databases that would require customer notification if breached or referenced in crisis simulations (see *[CMVM3.3 Simulate software crises]*).

[CP2.2: 38] 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, regardless of development methodology. The SSG might act as an advisor when the risk acceptor signs off on the state of the software prior to release. For example, the sign-off policy might require the head of the business unit to sign off on compliance issues that have not been mitigated or SSDL steps related to compliance that have been skipped. Sign-off should be explicit and captured for future reference. Any exceptions should be tracked, even under the fastest of agile methodologies. An application without security defects might still be noncompliant.

[CP2.3: 43] Implement and track controls for compliance.

The organization can demonstrate compliance with applicable regulations because its SSDL is aligned with the control statements developed by the SSG (see *[CP1.1 Unify regulatory pressures]*). The SSG tracks the controls, shepherds problem areas, and makes sure auditors and regulators are satisfied. If the organization's SDLC

is predictable and reliable, the SSG might be able to largely sit back and keep score. If the SDLC is uneven, less reliable, or trying to go faster than its supporting infrastructure can handle (looking at you, CI/CD), the SSG could be forced to take a more active role as referee. A firm doing this properly can explicitly associate satisfying its compliance concerns with following its SSDL.

[CP2.4: 42] Include software security SLAs in all vendor contracts.

Vendor contracts include an SLA ensuring that the vendor will not jeopardize the organization's compliance story and SSI. This is particularly important when selecting cloud computing providers. Each new or renewed contract contains a set of 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 further software security language in the SLA. Traditional IT security requirements and a simple agreement to allow penetration testing are not sufficient.

[CP2.5: 47] Ensure executive awareness of compliance and privacy obligations.

The SSG gains executive buy-in around compliance and privacy activities. Executives are provided plain-language explanations of the organization's compliance and privacy obligations, plus the potential consequences for failing to meet those obligations. For some organizations, explaining the direct cost and likely fallout from a compliance failure or data breach could be an effective way to broach the subject. For other organizations, having an outside expert address the Board works because some executives value outside perspective more than internal perspective. One sure sign of proper executive buy-in is adequate allocation of resources to get the job done. Be aware that the light and heat typically following a breach will not last.

CP LEVEL 3

[CP3.1: 21] Create a regulator compliance story.

The SSG has the information regulators want. 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 or piece of paper for every sprint. In some cases, regulators, auditors, and senior management are satisfied with the same kinds of reports, which might be generated directly from various tools. Governments are not the only regulators of behavior.

> The SSG can demonstrate the organization's compliance story without a fire drill.

[CP3.2: 12] 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. This goes for cloud and mobile platform providers as well. Evidence could include results from code reviews or penetration tests. Vendors may also attest to the fact that they are carrying out certain SSDL processes. In some cases, a BSIMM score or a vBSIMM score is used to help ensure that vendors are complying with the firm's policies.

[CP3.3: 5] Drive feedback from SSDL data back to policy.

Information from the SSDL is routinely fed back into the policy creation process to help find defects earlier or to

prevent them from occurring in the first place. Blind spots are eliminated based on trends in SSDL failures. For example, inadequate architecture analysis, recurring vulnerabilities, ignored security gates, or choosing the wrong firm to carry out a penetration test can expose policy weakness. Likewise, policies that impose too much bureaucracy might need to be adjusted to fit agile methodologies. Over time, policies should become more practical and easier to carry out (see *[SM1.1 Publish process (roles, responsibilities, plan), evolve as necessary]*). Ultimately, policies align themselves with the 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: 80] Provide awareness training.

The SSG provides awareness training in order to promote a culture of software security throughout the organization. Training might be delivered via SSG members, an outside firm, the internal training organization, or e-learning. Course content isn't necessarily tailored for a specific audience. For example, all programmers, QA engineers, and project managers could attend the same "Introduction to Software Security" course, but this activity should be enhanced with a tailored approach that addresses a firm's culture explicitly. Generic introductory courses that cover basic IT or high-level software security concepts do not generate satisfactory results. Likewise, awareness training aimed only at developers and not at other roles in the organization is insufficient.

[T1.5: 34] Deliver role-specific advanced curriculum (tools, technology stacks, and bug parade).

Software security training goes beyond building awareness by enabling trainees to incorporate security practices into their work. The training is tailored to cover the tools, technology stacks, development methodologies, and bugs that are most relevant to the trainee. An organization might offer four tracks for its engineers: one for architects, one for Java developers, one for mobile developers, and a fourth for testers. Tool-specific training is also commonly observed in a curriculum. Don't forget that training will be useful for many different roles in an organization, including QA, product management, executives, and others.

[T1.6: 26] 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 the problem, they are more likely to understand how the material is relevant to their work and to know when and how to apply what they have learned. One way to do this is to use noteworthy attacks on the company as examples in the training curriculum. Be wary of training that covers platforms not used by developers (Windows developers don't care about old Unix problems) or examples of problems only relevant to languages no longer in common use (Java developers don't need to understand buffer overflows in C). Stories from company history can help steer training in the right direction, but only if the stories are still relevant and not overly censored.

[T1.7: 47] Deliver on-demand individual training.

The organization lowers the burden on trainees 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 online courses must be engaging and relevant to achieve their intended purpose. Of course, training that sits around on the shelf does nobody any good, and hot topics like mobile and cloud will attract more interest than wonky policy discussions. For developers, it is possible to provide

training directly through the IDE right at the time that it's needed, but in some cases, building a new skill (such as code review) could be better suited for instructor-led training.

T LEVEL 2

[T2.5: 21] 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 AWS cloud development). Offering pizza and beer doesn't hurt. A standing conference call with voluntary attendance does not get the desired results, which are as much about building camaraderie as it is about sharing knowledge and organizational efficiency. There's no substitute for face-to-face meetings, even if they happen only once or twice a year.

[T2.6: 23] Include security resources in onboarding.

The process for bringing new hires into the engineering organization requires that they complete a training module about software security. The generic new hire process usually covers things like picking a good password and making sure that people don't tail you into the building, but this orientation period can be enhanced to cover topics such as secure coding, the SSDL, and internal security resources. The objective is to ensure that new hires contribute to the security culture. Turnover in engineering organizations is generally high, and although a generic onboarding module is useful, it does not take the place of a timely and more complete introductory software security course.

> Train everyone who works on your software, regardless of their employment status.

T LEVEL 3

[T3.1: 4] Reward progression through curriculum (certification or HR).

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 HR 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 HR 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.

[T3.2: 8] Provide training for vendors or outsourced workers.

Spending time and effort helping suppliers get security right at the outset is easier than trying to determine what went wrong later on, especially if the agile team has sprinted on to other projects. In the best case, outsourced workers receive the same training given to employees. Training individual contractors is much more natural than training entire outsource firms and is a reasonable place to start. Of course, it's important to train everyone who works on your software, regardless of their employment status.

[T3.3: 9] Host external software security events.

The organization highlights its security culture as a differentiator by hosting security events featuring external speakers and content. Good examples of this are Microsoft's BlueHat and QUALCOMM's Mobile Security Summit. Employees benefit from hearing outside perspectives, especially related to fast-moving technology areas. The organization as a whole benefits from putting its security cred on display (see *[SM3.2 Run an external marketing program]*). Events open to just certain small groups will not result in the desired change.

[T3.4: 9] Require an annual refresher.

Everyone involved in the SSDL is required to take an annual software security refresher course. This refresher keeps the staff up to date on security and ensures that the organization doesn't lose focus due to turnover, evolving methodologies, or changing deployment models. The SSG might use half a day to give an update on the security landscape and explain changes to policies and standards. A refresher can also be rolled out as part of a firm-wide security day or in concert with an internal security conference, but it is useful only if it's fresh.

> Make Sun Tzu proud by knowing your enemy.

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

The SSG offers help to any and all comers during an advertised lab 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. Office hours might be held one afternoon per week in the office of a senior SSG member. Roving office hours are also a possibility, with visits to particular product or application groups by request.

[T3.6: 3] Identify a satellite through training.

The satellite begins as a 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. Identifying this group proactively is a step toward creating a social network that speeds the adoption of security into software development. One way to begin is to track the people who stand out during training courses or office hours (see *[SM2.3 Create or grow a satellite]*). In general, a volunteer army may be easier to lead than one 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: 75] Create a data classification scheme and inventory.

The organization agrees on a data classification scheme and uses it to inventory its software according to the kinds of data the software handles, regardless of whether the software is 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 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, or 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 include generic sketches 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.

[AM1.5: 53] Gather and use attack intelligence.

The SSG stays ahead of the curve by learning about new types of attacks and vulnerabilities. The information comes from attending conferences and workshops, monitoring attacker forums, and reading relevant publications, mailing lists, and blogs. Make Sun Tzu proud by knowing your enemy; engage with the security researchers who are likely to cause you trouble. In many cases, a subscription to a commercial service provides a reasonable way of gathering basic attack intelligence. Regardless of its origin, attack information must be made actionable and useful for software builders and testers.

> Don't just build an attack list; use it.

AM LEVEL 2

[AM2.1: 10] Build attack patterns and abuse cases tied to potential attackers.

The SSG prepares for security testing and architecture analysis by building attack patterns and abuse cases tied to potential attackers (see *[AM1.3 Identify potential attackers]*). These resources don't have to be built from scratch for every application to be useful. Instead, there could be standard sets for applications with similar profiles. The SSG will add to the pile based on attack stories. For example, a story about an attack against a poorly designed cloud application could lead to a cloud security attack pattern that drives a new type of testing. If a firm tracks fraud and monetary costs associated with particular attacks, this information can be used to prioritize the process of building attack patterns and abuse cases.

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

The SSG creates technology-specific attack patterns to capture knowledge about attacks that target particular technologies. For example, if the organization's cloud software relies on the cloud vendor's security apparatus (e.g., cryptography), the SSG could catalogue the quirks of the crypto package and how it might be exploited. Attack patterns directly related to the security frontier (e.g., IoT) can be useful as well. Simply republishing a general guideline (e.g., "Ensure data are protected in transit") and adding "for mobile applications" on the end does not constitute a technology-specific attack pattern.

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

The SSG helps the organization understand attack basics by maintaining a living list of important attacks and using it to drive change. This list combines input from multiple sources such as observed attacks, hacker forums, industry trends, new technology stacks or deployment methods in use, etc. It does not need to be updated with great frequency, and attacks can be coarsely sorted. For example, the SSG might brainstorm twice per year to create lists of attacks the organization should be prepared to counter "now," "soon," and

"someday." In some cases, attack model information is used in a list-based approach to architecture analysis, helping focus the analysis as in the case of *STRIDE*. Don't just build the list; use it.

[AM2.6: 14] 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. 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's overly focused on top 10 lists or irrelevant and outdated platform attacks (see *[T1.6 Create and use material specific to company history]*). Hiding information about attacks from people building new systems does nothing to garner positive benefit from a negative happenstance.

[AM2.7: 11] Build an internal forum to discuss attacks.

The organization has an internal forum where the SSG, the satellite, and others discuss attacks and attack methods. The forum serves to communicate the attacker perspective. The SSG could also maintain an internal mailing list where subscribers 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 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. Vigilance means never getting too comfortable (see *[SR1.2 Create a security portal]*).

AM LEVEL 3

[AM3.1: 4] Have a science team that develops new attack methods.

The SSG has a science team that works to identify and defang new classes of attacks before real attackers even know that they exist. Because the security implications of new technologies have not been fully explored in the wild, doing it yourself 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. A science team may include well-known security researchers who publish their findings at conferences like *DEF CON*.

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

The SSG arms testers with automation to mimic what attackers are going to do. For example, a new attack method identified by the science team could require a new tool, so the SSG packages the new tool and distributes it to testers. The idea here is to push attack capability past what typical commercial tools and offerings encompass, and then repurpose that information for others to use. Tailoring these new tools to a firm's particular technology stacks and potential attackers is a good idea as well. When technology stacks and coding languages are evolving faster than vendors can innovate, creating your own tools might be the best way forward.

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: 95] Build and publish security features.

Some problems are best solved only once. Rather than have each project team implement all of their own security features (e.g., authentication, role management, key management, audit/log, cryptography, protocols), the SSG provides proactive guidance by building and publishing security features for other groups to use. Generic security features often have to be tailored for specific platforms, such as mobile. For example, a mobile crypto feature will need at least two versions to cover Android and iOS if it uses low-level system calls. 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. The SSG can identify an implementation that it likes and promote it as the accepted solution.

Proactive engagement by the SSG is key to success. Assume nothing.

[SFD1.2: 70] Engage SSG with architecture.

Security is a regular topic in the organization's software architecture discussion, and the architecture group takes responsibility for security the same way in which it takes responsibility for performance, availability, or scalability. One way to keep security from falling out of this discussion is to have an SSG member attend regular architecture meetings. In other cases, enterprise architecture can help the SSG create secure designs that integrate properly into corporate design standards. Proactive engagement by the SSG is key to success. Moving a well-known system to the cloud means reengaging the SSG, for example. Assume nothing.

SFD LEVEL 2

[SFD2.1: 34] Build secure-by-design middleware frameworks and common libraries.

The SSG takes a proactive role in software design by building or providing pointers to secure-by-design middleware frameworks or common libraries. In addition to teaching by example, this middleware aids architecture analysis and code review because the building blocks make it easier to spot errors. For example, the SSG could modify a popular web framework, such as Spring, to make it easy to meet input validation requirements. Eventually, the SSG can tailor code review rules specifically for the components it offers (see *[CR3.1 Use automated tools with tailored rules]*). When adopting a middleware framework (or any other widely used software), careful vetting for security before publication is important. Encouraging adoption and use of insecure middleware does not help the software security situation. Generic open source software security architectures, including OWASP ESAPI, should not be considered secure by design. Bolting security on at the end by calling a library is not the way to approach secure design.

[SFD2.2: 46] Create SSG capability to solve difficult design problems.

When the SSG is involved early in a new project process, it contributes to new architecture and solves difficult design problems, minimizing the negative impact that security has on other constraints (time to market, price, etc.). If a skilled security architect from the SSG is involved in the design of a new protocol, he or she could analyze the security implications of existing protocols and identify elements that should be 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 then refactoring when flaws are uncovered. Of course, even the best expert might not scale to cover the needs of an entire software portfolio. Some design problems will require specific expertise outside of the SSG.

SFD LEVEL 3

[SFD3.1: 9] Form a review board or central committee to approve and maintain secure design patterns.

A review board or central committee formalizes the process for reaching consensus on design needs and security tradeoffs. Unlike the architecture committee, this group is specifically focused on providing security guidance and also periodically reviews already published design standards (especially around authentication, authorization, and cryptography) to ensure that design decisions do not become stale or out of date. Moreover, a review board can help control the chaos often associated with the adoption of new technologies when development groups might otherwise head out into the wild on their own without ever engaging the SSG.

[SFD3.2: 9] Require use of approved security features and frameworks.

Implementers take their security features and frameworks from an approved list. There are two benefits to this activity: developers do not spend time reinventing existing capabilities, and review teams do not have to contend with finding the same old defects in brand new projects or when new platforms are adopted. In particular, the more a project uses proven components, the easier the architecture analysis and code review 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.

[SFD3.3: 2] Find and publish mature design patterns from the organization.

The SSG fosters centralized design reuse by collecting design patterns from across the organization and publishing them for everyone to use. A section of the SSG website could promote positive elements identified during architecture analysis so that good ideas are spread. This process should be formalized: an ad hoc, accidental noticing is not sufficient. In some cases, a central architecture or technology team can facilitate and enhance this activity. Common design patterns make development faster, such as using secure design patterns for all software, not just applications (microservices, APIs, frameworks, infrastructure, and automation).

> Standards that are not widely adopted and enforced are not really standards.

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: 75] Create security standards.

Software security requires much more than security features, but security features are part of the job as well. The SSG meets the organization's demand for security guidance by creating standards that explain the

accepted way to adhere to policy and carry out specific security-centric operations. A standard might describe how to perform authentication on an Android device or how to determine the authenticity of a software update (see *[SFD1.1 Build and publish security features]* for one case where the SSG provides a reference implementation of a security standard). Often, software that is not an application requires its own standard (e.g., an API or a microservices architecture). Standards can be deployed in a variety of ways. In some cases, standards and guidelines can be automated in development environments (e.g., worked into an IDE), but in others, guidance can be explicitly linked to code examples or even containers to make them more actionable and relevant. Standards that are not widely adopted and enforced are not really standards.

[SR1.2: 78] 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 and face-to-face meetings.

[SR1.3: 76] Translate compliance constraints to requirements.

Compliance constraints are translated into software requirements for individual projects. This is a linchpin in the organization's compliance strategy: by representing compliance constraints explicitly with requirements, 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 could play a role in the SSDL during the requirements phase. In other cases, technology standards built for international interoperability can include security guidance. Representing these standards as requirements helps with traceability and visibility in the event of an audit. It's particularly useful to codify the requirements in reusable code or containers.

SR LEVEL 2

[SR2.2: 38] Create a standards review board.

The organization creates a standards review board to formalize the process used to develop standards and ensure that all stakeholders have a chance to weigh in. The 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.

[SR2.3: 23] Create standards for technology stacks.

The organization standardizes on specific technology stacks. For the SSG, this means a reduced workload because the group does not have to explore new technology risks for every new project. Ideally, the organization will create a secure base configuration for each technology stack, further reducing the amount of work required to use the stack safely. A stack might include an operating system, a database, an application server, and a runtime environment for a managed language. The security frontier is a good place to find traction. Currently, mobile technology stacks and platforms as well as cloud-based technology stacks are areas where specific attention to security particularly pays off.

[SR2.4: 39] Identify open source.

The first step toward managing the risk introduced by open source is to identify the open source components in use across the portfolio and really understand the dependencies. It's not uncommon to discover old versions of components with known vulnerabilities or 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. An informal annual review or a process that relies solely on developers asking for permission does not generate satisfactory results. At the next level of maturity, this activity is subsumed by a policy constraining the use of open source.

Guidance does not a standard make.

[SR2.5: 29] Create SLA boilerplate.

The SSG works with the legal department to create standard SLA boilerplate that is used 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 software security vendor management solutions, such as vBSIMM measurements or BSIMM scores.

SR LEVEL 3

[SR3.1: 17] Control open source risk.

The organization has control over its exposure to the vulnerabilities that come along with using open source components and their army of dependencies. Use of open source could be restricted to predefined projects or to open source versions that have been through an SSG security screening process, had unacceptable vulnerabilities remediated, and are made available only through internal repositories. The legal department often spearheads additional open source controls due to the "viral" license problem associated with GPL code. In general, getting the legal department to understand security risks can help move an organization to improve its open source practices. Of course, this control must be applied across the software portfolio.

[SR3.2: 10] 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 cannot be guaranteed through contract language alone. The SSG engages with vendors, discusses vendor security practices, and explains 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 win. When a firm's SSDL is available publicly, communication regarding software security expectations is easier. Likewise, sharing internal practices and measures can make expectations clear. Don't work with a vendor that has worse security policies than you do.

[SR3.3: 10] Use secure coding standards.

Secure coding standards help developers avoid the most obvious bugs and provide ground rules for code review. Secure coding standards are necessarily specific to a programming language or platform, and they can address the use of popular frameworks and libraries, but mobile platforms need their own specific coding standards. If the organization already has coding standards for other purposes, the 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 beef up security training with relevant examples. Remember, guidance does not a standard make.

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: 101] Perform security feature review.

To get started in architecture analysis, center the process on a review of security features. Security-aware reviewers identify the security features in an application (authentication, access control, use of cryptography, etc.) and then study the design looking for problems that would cause these features to fail at their purpose or otherwise prove insufficient. For example, a system that was subject to escalation of privilege attacks because of broken access control or a mobile application that stashed away PII on local storage would both be identified in this kind of review. At higher levels of maturity, the activity of reviewing features is eclipsed by a more thorough approach to AA. In some cases, use of the firm's secure-by-design components can streamline this process.

[AA1.2: 33] Perform design review for high-risk applications.

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

[AA1.3: 27] 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 the architects, and proficiency requires practice. The SSG cannot 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 carry out the detailed review with a minimum of interaction with the project team. At higher levels of maturity, the responsibility for leading review efforts shifts toward software architects. Approaches to AA, including threat modeling, evolve over time, so it is wise to not expect to set a process and use it forever.

> Do not expect to set a process and use it forever.

[AA1.4: 57] Use a risk questionnaire to rank applications.

To facilitate security feature and design review processes, the SSG uses a risk questionnaire to collect basic information about each application so it can determine a risk classification and prioritization scheme. Questions might include, "Which programming languages is the application written in?", "Who uses the application?", and "Is the application deployed in a container?" A qualified member of the application team completes the questionnaire, which should be short enough that it

can be completed in a matter of minutes. The SSG might use the answers to categorize the application as 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 impotent.

AA LEVEL 2

[AA2.1: 15] 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, security properties, and the associated risk, and it is defined rigorously enough that people outside the SSG can be taught to carry it out. Particular attention should be paid to documentation of both the architecture under review and any security flaws uncovered. Tribal knowledge doesn't count as a defined process. Microsoft's STRIDE and Synopsys' ARA are examples of this process, although even these two methodologies for AA have evolved greatly over time.

[AA2.2: 14] Standardize architectural descriptions (including data flow).

Defined AA processes (see *[AA2.1 Define and use AA process]*) use an agreed-upon format to describe architecture, including a means for representing data flow. This format, combined with a documented AA process, makes AA tractable for people who are not security experts. In the case of cloud applications, data are likely to flow across the Internet, so a network diagram is useful in this case, 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. Standardized icons that are consistently used in UML diagrams, Visio templates, and whiteboard squiggles are especially useful, too.

AA LEVEL 3

[AA3.1: 4] Have software architects lead design review efforts.

Software architects throughout the organization lead the AA process most of the time. Although the SSG still might contribute to AA in an advisory capacity or under special circumstances, this activity requires a well-understood and well-documented process (see *[AA2.1 Define and use AA process]*). Even then, consistency is difficult to attain because breaking architecture requires experience.

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

Failures identified during AA are fed back to the security design committee 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]*). Security design patterns can interact in surprising ways that break security. The AA process should be applied even when vetted design patterns are in standard use.

[AA3.3: 3] 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 who ask for help in using the AA process (see *[AA2.1 Define and use AA process])* to conduct their own design review. The SSG will 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 plays 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: 82] Have the SSG perform ad hoc review.

The SSG performs an ad hoc code review for high-risk applications in an opportunistic fashion, such as by following up the design review for high-risk applications with a code review. At higher maturity levels, this informal targeting is replaced with a systematic approach. SSG review could involve the use of specific tools and services, or it might be manual, but it has to be proactive. When new technologies pop up, new approaches to code review might become necessary.

> Individual bugs make excellent training examples.

[CR1.4: 76] Use automated tools along with manual review.

Incorporate static analysis into the code review process to make code review more efficient and more consistent. The automation doesn't replace human judgment, but it does bring definition to the review process and security expertise to reviewers who are not security experts. Note that a specific tool might not cover an entire portfolio, especially when new languages are involved, but that's no excuse not to review the code. A firm may use an external service vendor as part of a formal code review process for software security, and this service should be explicitly connected to a larger SSDL applied during software development, not just used to "check the security box" on the path to deployment.

[CR1.5: 40] Make code review mandatory for all projects.

Code review is a mandatory release gate for all projects under the SSG's purview. Lack of code review or unacceptable results will stop a release or slow it down. While all projects must undergo code review, the review 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. In most cases, a code review gate with a minimum acceptable standard forces projects that don't pass to be fixed and reevaluated before they ship. A code review tool with nearly all the rules turned off so it can run at CI/CD automation speeds won't provide sufficient defect coverage.

[CR1.6: 44] 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. Code review information can be incorporated into a CISO-level dashboard that includes feeds from other parts of the security organization (e.g., penetration tests, security testing, black-box testing, and white-box testing). 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.

CR LEVEL 2

[CR2.5: 28] Assign tool mentors.

Mentors are available to show developers how to get the most out of code review tools. If the SSG is most skilled with the tools, it could use office hours to help developers establish the right configuration or get started 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 over time through the use of tool mentors. Providing installation instructions and URLs to centralized tools does not count as mentoring.

[CR2.6: 20] Use automated tools with tailored rules.

Customize static analysis to improve efficiency and reduce false positives. Use custom rules to find errors specific to the organization's coding standards or custom middleware. Turn off checks that aren't relevant. 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 code base in a negative sense.

[CR2.7: 25] Use a top *N* bugs list (real data preferred).

The SSG maintains a list of the most important kinds of bugs that it wants to eliminate from the organization's code and uses it to drive change. It's okay to start with a generic list pulled from public sources, but a list is much more valuable if it's specific to the organization and built from real data gathered from code review, testing, software composition analysis, and actual incidents. The SSG can periodically update the list and publish a "most wanted" report. (For another way to use the list, see *[T1.6 Create and use material specific to company history]*). Some firms use multiple tools and real code base data to build top *N* lists, not constraining themselves to a particular service or tool. One potential pitfall with a top *N* list is the problem of "looking for your keys only under the street light"—that is, it only includes known problems. For example, the OWASP Top 10 list rarely reflects an organization's bug priorities. Simply sorting the day's bug data by number of occurrences doesn't produce a satisfactory top *N* list because these data change so often. A top *N* bugs list should be used to kill bugs.

A top *N* bugs list should be used to kill bugs.

CR LEVEL 3

[CR3.2: 4] Build a factory.

Combine assessment results so that multiple analysis techniques feed into one reporting and remediation process. The SSG might write scripts to invoke multiple detection techniques automatically and combine the results into a format that can be consumed by a single downstream review and reporting solution. Analysis engines may combine static and dynamic analysis, and different review streams, such as mobile versus standard approaches, can be unified with a factory. The tricky part of this activity is normalizing vulnerability information from disparate sources that use conflicting terminology. In some cases, using a standardized taxonomy (perhaps a CWE-like approach) can help with normalization. Combining multiple sources helps drive better-informed risk mitigation decisions.

[CR3.3: 1] Build a capability for eradicating specific bugs from the entire codebase.

When a new kind of bug is found, the SSG writes rules to find it and uses the rules to identify all occurrences of the new bug throughout the entire codebase. It's 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 will have an easier time with this activity than firms with a large number of large applications.

[CR3.4: 4] 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 back doors, 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 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 it is insufficient to complete this activity.

[CR3.5: 3] Enforce coding standards.

A violation of the organization's secure coding standards is sufficient grounds for rejecting a piece of code. Code review is objective—it shouldn't devolve into a debate about whether or not bad code is exploitable. The enforced portion of the standard could start out being as simple as a list of banned functions. In some cases, coding standards for developers are published specific to technology stacks (for example, guidelines for C++, Spring, or Swift) and then enforced during the code review process or directly in the IDE. Standards can be positive ("do it this way") or negative ("do not use this API").

SSDL TOUCHPOINTS: Security Testing (ST)

The Security Testing practice is concerned with prerelease testing, including integrating security into standard QA processes. The practice includes 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: 100] Ensure QA supports edge/boundary value condition testing.

The QA team goes beyond functional testing to perform basic adversarial tests and probe simple edge cases and boundary conditions, no attacker skills required. When QA understands the value of pushing past standard functional testing using acceptable input, it begins to move slowly toward thinking like an adversary. A discussion of boundary value testing leads naturally to the notion of an attacker probing the edges on purpose. What happens when you enter the wrong password over and over?

[ST1.3: 88] Drive tests with security requirements and security features.

Testers target declarative security mechanisms with tests derived from requirements and security features. A tester 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 based on requirements such as account lockout, transaction limitations, entitlements, and so on are also tested. Of course, software security is not security software, but getting started with features is easy. New deployment models, such as cloud, might require novel test approaches.

PT LEVEL 2

[ST2.1: 30] 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; tools such as IBM Security AppScan or Fortify WebInspect are relevant for web applications, and fuzzing frameworks such as Synopsys Defensics are applicable for most network protocols. 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. Because of the way testing is integrated into agile development approaches, black-box tools might be used directly by engineering. Regardless of who runs the black-box tool, the testing should be properly integrated into the QA cycle of the SSDL.

[ST2.4: 14] Share security results with QA.

The SSG routinely shares results from security reviews with the QA department. Using security results to inform and evolve particular testing patterns can be a powerful mechanism leading to better security testing. CI/CD makes this easier because of the way testing is integrated in a cross-functional team. Over time, QA engineers learn the security mindset, and this activity benefits from an engineering-focused QA function that is highly technical.

> Software security is not security software.

[ST2.5: 12] Include security tests in QA automation.

Security tests run alongside functional tests as part of automated regression testing. In fact, the same automation framework houses both, and security testing is part of the routine. Security tests can be driven from abuse cases identified earlier in the lifecycle or tests derived from creative tweaks of functional tests.

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

Test automation engineers or agile team members customize a fuzzing framework to the organization's APIs. They could begin from scratch or use an existing fuzzing toolkit, but customization goes beyond creating custom protocol descriptions or file format templates. The fuzzing framework has 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: 4] 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 architecture analysis concludes 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.

[ST3.4: 3] 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. 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 is not a testing best practice. Using standard measurements for coverage such as function coverage, line coverage, or multiple condition coverage is fine.

[ST3.5: 3] Begin to build and apply adversarial security tests (abuse cases).

Testing begins to incorporate test cases based on abuse cases (see *[AM2.1 Build attack patterns and abuse cases tied to potential attackers]*), and 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 driven from security policies, attack intelligence, and standards. This turns the corner from testing features to attempting to break the software under test.

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: 105] Use external penetration testers to find problems.

Many organizations aren't willing to address software security until there's unmistakable evidence that the organization isn't somehow magically immune to the problem. If security has not been a priority, external penetration testers can demonstrate that the organization's code needs help. Penetration testers could be brought in to break a high-profile application to make the point. Over time, the focus of penetration testing moves from "I told you our stuff was broken" to a smoke test and sanity check done before shipping. External penetration testers bring a new set of eyes to the problem.

> If your penetration tester doesn't ask for the code, you need a new penetration tester.

[PT1.2: 89] Feed results to the defect management and mitigation system.

Penetration testing results are fed back to development through established defect management or mitigation channels, and development responds via a defect management and release process. Emailing them around doesn't count. Properly done, the exercise demonstrates the organization's ability to improve the state of security, and many firms are beginning to emphasize 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. Evolving DevOps and integrated team structures do not eliminate the need for formalized defect management systems.

[PT1.3: 74] Use penetration testing tools internally.

The organization creates an internal penetration testing capability that uses tools. This capability can be part of the SSG or part of a specialized team elsewhere in the organization, with the tools improving the efficiency and repeatability of the testing process (and frequently being a necessary part of CI/CD environments). Tools can include off-the-shelf products, standard-issue network penetration tools that understand the application layer, and handwritten scripts. Free-time or crisis-driven efforts do not constitute an internal capability.

PT LEVEL 2

[PT2.2: 26] Provide penetration testers with all available information.

Penetration testers, whether internal or external, can do deeper analysis and find more interesting problems after they receive source code, design documents, architecture analysis results, and code review results. Penetration testers need everything that is created throughout the SSDL. If your penetration tester doesn't ask for the code, you need a new penetration tester.

[PT2.3: 21] Schedule periodic penetration tests for application coverage.

The SSG periodically tests all applications in its purview according to an established schedule, which could be tied to a calendar or a release cycle. High-profile applications might get a penetration test at least once a year. This testing serves as a sanity check and helps ensure that yesterday's software isn't vulnerable to today's attacks; it also helps 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.

> Doing software security before network security is like putting on pants before putting on underwear.

PT LEVEL 3

[PT3.1: 10] 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 are 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 prevents checklist-driven approaches that only look for known types of problems; it's pretty much essential when it comes to new technology.

[PT3.2: 7] Have the SSG customize penetration testing tools and scripts.

The SSG either creates penetration testing tools or adapts publicly available ones to more efficiently and comprehensively attack the organization's systems. Tools improve the efficiency of the penetration testing process without sacrificing the depth of problems that the SSG can identify. Automation can be particularly valuable under agile methodologies because it helps teams go faster. Tools that can be tailored are always preferable to generic tools. This activity considers 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), web application firewalls, installation and configuration documentation, containerization, orchestration, application monitoring, change management, and code signing.

SE LEVEL 1

[SE1.1: 58] Use application input monitoring.

The organization monitors the input to the software that it runs in order to spot attacks. For web code, a web application firewall (WAF) can do the job; other kinds of software likely require other approaches. The SSG might be responsible for the care and feeding of the system, but incident response is not part of this activity. Defanged WAFs that write log files can be useful if someone periodically reviews the logs. A WAF that's unmonitored makes no noise when an application falls in the woods.

[SE1.2: 104] Ensure host and network security basics are in place.

The organization provides a solid foundation for software by ensuring that host and network security basics are in place. Operations security teams are usually responsible for patching operating systems, maintaining firewalls, and properly configuring cloud services, but doing software security before network security is like putting on pants before putting on underwear.

SE LEVEL 2

[SE2.2: 39] Publish installation guides.

The SSDL requires the creation of an installation guide or a clearly described configuration, such as for a container, to help deployment teams and operators install and configure the software securely. If special steps are required to ensure a deployment is secure, the steps are either outlined in the installation guide or explicitly noted in deployment automation. The guide should include a discussion of COTS components, too. In some cases, installation guides are distributed to customers who buy the software. Make sure that all deployment automation can be understood by smart humans and not just by a machine. Evolving DevOps and integrated team structures do not eliminate the need for human-readable guidance. Of course, secure by default is always the best way to go.

[SE2.4: 31] Use code signing.

The organization uses code signing for software published across trust boundaries. Code signing is particularly useful for protecting the integrity of software that leaves the organization's control, such as shrink-wrapped applications or thick clients. The fact that some mobile platforms require application code to be signed does not indicate institutional use of code signing.

SE LEVEL 3

[SE3.2: 17] Use code protection.

To protect intellectual property and make exploit development harder, the organization erects barriers to reverse engineering. This is particularly important for widely distributed mobile applications. Obfuscation techniques could be applied as part of the production build and release process. Employing platform-specific controls such as Data Execution Prevention (DEP), Safe Structured Error Handling (SafeSEH), and Address Space Layout Randomization (ASLR) can make exploit development more difficult.

[SE3.3: 4] Use application behavior monitoring and diagnostics.

The organization monitors the behavior of 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. Intrusion detection and anomaly detection systems at the application level may 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.

[SE3.4: 11] Use application containers.

The organization uses application containers to support its software security goals. The primary drivers for using containers include ease of deployment, a tighter coupling of applications with their dependencies, and isolation without the overhead of deploying a full OS on a virtual machine. Containers provide a convenient place for security controls to be applied and updated consistently. The ones used in development or test environments without reference to security do not count.

[SE3.5: 0] Use orchestration for containers and virtualized environments.

The organization uses automation to scale container and virtual machine deployments in a disciplined way. Orchestration processes take advantage of built-in and add-on security controls to ensure each deployed container and virtual machine meets predetermined security requirements. Setting security behaviors in aggregate allows for rapid change when the need arises. Of course, orchestration platforms are themselves software that, in turn, requires security patching and configuration. If you use Kubernetes, make sure you patch Kubernetes.

[SE3.6: 0] 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 applications]*). In addition, a manifest detailing the components, dependencies, configurations, external services, and so on for all production software allows organizations to secure all the things. 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 all the components 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.

[SE3.7: 0] Ensure cloud security basics.

Of course, you already do *[SE1.2 Ensure host and network security basics are in place]*, right? Someone must ensure that basic requirements are met in cloud deployments as well. In the increasingly software-defined world, you must explicitly implement security features and controls (some of which may be built in) at least as good as those built with cables and physical hardware. Nothing is as automatic as it seems.

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: 101] Create or interface with incident response.

The SSG is prepared to respond to an incident and is regularly included in the incident response process, either by creating its own incident response capability or 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. Sometimes cloud service providers need to be looped in as well. In many cases, SSIs evolved from incident response teams who began to realize that software vulnerabilities were the bane of their existence.

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

Defects identified 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). In some cases, providing a way to enter incident triage data into an existing bug-tracking system (perhaps making use of a special security flag) seems to work. The idea is to close the information loop and make sure that security problems get fixed. In the best of cases, processes in the SSDL can be improved based on operational data.

CMVM LEVEL 2

[CMVM2.1: 82] Have emergency codebase response.

The organization can make quick code changes when an application is under attack. A rapid-response team works in conjunction with the application owners and the SSG to study the code and the attack, find a resolution, and push a patch into production. Often, the emergency response team is the development team itself, especially when agile methodologies are in use. Fire drills don't count; a well-defined process is required, and a process that has never been used might not actually work.

[CMVM2.2: 87] Track software bugs found in operations through the fix process.

Defects found in operations are fed back to development, 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 the bug finders and the bug fixers. Make sure the loop is closed completely. Setting a security flag in the bug-tracking system can help facilitate tracking.

[CMVM2.3: 57] Develop an operations inventory of applications.

The organization has a map of its software deployments. If a piece of code needs to be changed, Operations or DevOps can reliably identify all the places where the change needs to be installed. Common components shared between multiple projects are noted so that, when an error occurs in one application, other applications that share the same components can be fixed as well. Remember, open source components are components, too.

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. This requires the ability to reexamine the entire codebase when new kinds of bugs come to light (see *[CR3.3 Build capability for eradicating specific bugs from entire codebase]*). One way to approach this is to create a rule set that generalizes a deployed bug into something that can be scanned for via automated code review.

[CMVM3.2: 7] Enhance the SSDL to prevent software bugs found in operations.

Experience from operations leads to changes in the SSDL, which is strengthened to prevent the reintroduction of bugs found during operations. To make this process systematic, each incident response postmortem could include a "feedback to SSDL" step. This works best when root-cause analysis pinpoints where in the SDLC an error could have been introduced or slipped by uncaught. Cross-functional DevOps teams might have an easier time with this because all the players are involved. An ad hoc approach is not sufficient.

An ad hoc approach is not sufficient.

[CMVM3.3: 9] Simulate software crises.

The SSG simulates high-impact software security crises to ensure software incident response capabilities minimize damage. Simulations could test for the ability to identify and mitigate specific threats or, in other cases, could begin with the assumption that a critical system or service is already compromised and evaluate the organization's ability to respond. 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. If the data center is burning to the ground, the SSG won't be among the first responders.

[CMVM3.4: 13] 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, do not count.

APPENDIX

Adjusting BSIMM8 for BSIMM9

Because the BSIMM is a data-driven model, we have chosen to make adjustments to the model based on the data observed between BSIMM8 and BSIMM9.

We have added, deleted, and adjusted the levels of various activities based on the data observed as the study continues. To preserve backward compatibility, all changes are made by adding new activity labels to the model, even when an activity has simply changed levels. We make changes by considering outliers both in the model itself and in the levels we assigned in the 12 practices. We use the results of an intralevel standard deviation analysis to determine which outlier activities to move between levels, focusing on changes that minimize standard deviation in the average number of observed activities at each level.

Here are the five changes we made according to that paradigm:

1. [SM2.5 Identify metrics and use them to drive budgets] became SM3.3
2. [SR2.6 Use secure coding standards] became SR3.3
3. [SE3.5 Use orchestration for containers and virtualized environments] added to the model
4. [SE3.6 Enhance application inventory with operations bill of materials] added to the model
5. [SE3.7 Ensure cloud security basics] added to the model

We also carefully considered, but did not adjust *[T1.6 Create and use material specific to company history]*.

The activities that are now SM3.3 and SR3.3 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 moved to SM3.3. The BSIMM1 activity *[SR1.4 Use coding standards]* became SR2.6 in BSIMM6 and is now moved to SR3.3.

We noted in BSIMM7 that, for the first time, one activity *[AA3.2 Drive analysis results into standard architecture patterns]*, was not observed in the current data set, and there were no new observations of AA3.2 for BSIMM8. AA3.2 does have two observations in BSIMM9, and there are no activities with zero observations except for the three just added.

One question that recently came up is, "Where do activities go to die?" We've noticed that a handful of activities have moved from level 1 through level 2 to level 3 for all the wrong reasons. These activities may disappear in future BSIMM iterations. The two most prominent contenders are *[T3.5 Establish SSG office hours]* and *[T3.6 Identify a satellite through training]*, both of which appear to be going extinct. Less pronounced, but still worth noting, are *[SM3.3 Identify metrics and use them to drive budgets] and [SR3.3 Use secure coding standards].*

116 BSIMM Activities at a Glance

(**Red** indicates most observed BSIMM activity in that practice)

Level 1 Activities

Governance

Strategy & Metrics (SM)

- Publish process (roles, responsibilities, plan), evolve as necessary. [SM1.1]
- Create evangelism role and perform internal marketing. [SM1.2]
- Educate executives. [SM1.3]
- **Identify gate locations, gather necessary artifacts. [SM1.4]**

Compliance & Policy (CP)

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

Training (T)

- **Provide awareness training. [T1.1]**
- Deliver role-specific advanced curriculum (tools, technology stacks, and bug parade). [T1.5]
- Create and use material specific to company history. [T1.6]
- Deliver on-demand individual training. [T1.7]

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)

- **Build and publish security features. [SFD1.1]**
- Engage SSG with architecture. [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 questionnaire to rank applications. [AA1.4]

Code Review (CR)

- **Have SSG perform ad hoc 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]

Security Testing (ST)

- **Ensure QA supports 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. [CMVM 1.2]**

Level 2 Activities

Governance

Strategy & Metrics (SM)

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

Compliance & Policy (CP)

- Identify PII data 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]
- Include security resources in onboarding. [T2.6]

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)

- Build secure-by-design middleware frameworks and common libraries. [SFD2.1]
- Create SSG capability to solve difficult design problems. [SFD2.2]

Standards & Requirements (SR)

- Create a standards review board. [SR2.2]
- Create standards for technology stacks. [SR2.3]
- Identify open source. [SR2.4]
- Create a SLA boilerplate. [SR2.5]

SSDL Touchpoints

Architecture Analysis (AA)

- Define and use AA process. [AA2.1]
- Standardize architectural descriptions (including data flow). [AA2.2]

Code Review (CR)

- Assign tool mentors. [CR2.5]
- 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)

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

Software Environment (SE)

- Publish installation guides. [SE2.2]
- Use code signing. [SE2.4]

Configuration Management & Vulnerability Management (CMVM)

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

Level 3 Activities

Governance

Strategy & Metrics (SM)

- Use an internal tracking application with portfolio view. [SM3.1]
- Run an external marketing program. [SM3.2]
- Identify metrics and use them to drive budgets. [SM3.3]

Compliance & Policy (CP)

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

Training (T)

- Reward progression through curriculum (certification or HR). [T3.1]
- Provide training for vendors or outsourced workers. [T3.2]
- Host external software security events. [T3.3]
- Require an annual refresher. [T3.4]
- Establish SSG office hours. [T3.5]
- Identify a satellite through training. [T3.6]

Intelligence

Attack Models (AM)

- Have a science team that develops new attack methods. [AM3.1]
- Create and use automation to mimic attackers. [AM3.2]

Security Features & Design (SFD)

- Form a review board or central committee to approve and maintain secure design patterns. [SFD3.1]
- Require use of approved security features and frameworks. [SFD3.2]
- Find and publish mature 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]

SSDL Touchpoints

Architecture Analysis (AA)

- Have software architects lead design review efforts. [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 factory. [CR3.2]
- Build a capability for eradicating specific bugs from the entire codebase. [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]

Deployment

Penetration Testing (PT)

- Use external penetration testers to perform deep-dive analysis. [PT3.1]
- Have the SSG customize penetration testing tools and scripts. [PT3.2]

Software Environment (SE)

- Use code protection. [SE3.2]
- Use application behavior monitoring and diagnostics. [SE3.3]
- Use application containers. [SE3.4]
- Use orchestration for containers and virtualized environments. [SE3.5]
- Enhance application inventory with operations bill of materials. [SE3.6]
- Ensure cloud security basics. [SE3.7]

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]

Interested in joining the growing BSIMM Community?

Go to ***www.BSIMM.com***