Building an Enterprise DevSecOps Program

November 26, 2019

Securosis, L.L.C. 515 E. Carefree Highway Suite #766 Phoenix, AZ 85085 **T** 602-412-3051
info@securosis.com www.securosis.com

Author's Note

The content in this report was *developed independently of any licensees*. It is based on material originally posted on the Securosis blog, but has been enhanced, reviewed, and professionally edited.

Special thanks to Chris Pepper for editing and content support.

This report licensed by Veracode.

With its combination of automation, process, and speed, Veracode becomes a seamless part of the software lifecycle, eliminating the friction that arises when security is detached from the development and deployment process. As a result, enterprises are able to fully realize the advantages of DevOps environments while ensuring secure code is synonymous with high-quality code.

Veracode serves more than 2,300 customers worldwide across a wide range of industries. The Veracode Platform has assessed more than 10 trillion lines of code and helped companies fix 36 million security flaws to date.

Learn more at www.veracode.com.

Copyright

Table of Contents

Introduction

DevOps is an operational framework which promotes software consistency and standardization through automation. It helps address many nightmare development issues around integration, testing, patching, and deployment — both by breaking down barriers between different development teams, and also by prioritizing things which make software development simpler, faster and easier.

DevSecOps is the integration of security teams and security tools directly into the software development lifecycle, leveraging the automation and efficiencies of DevOps to ensure application security testing occurs in every build cycle. This promotes security and consistency, and helps to ensure that security is prioritized no lower than other quality metrics or features. Automated security testing, just like automated application build and deployment, must be assembled with the rest of the infrastructure.

The Elephant in the Room: DevOps vs. DevSecOps

Which leads us to a controversial topic, and why this research is different: the name DevSecOps. We contend that calling out security — the 'Sec' in 'DevSecOps' — is needed in light of maturity and understanding of this topic. In our 2015 work "Building Security Into DevOps" we embraced the idea that security was an equal partner and there was no reason to call out security specifically. In hindsight, this was wrong. The fact is security practitioners are having a much harder DevOps journey, and they are the ones struggling, and they are the ones who need a roadmap on security integration.

Stated another way, practitioners of DevOps who have fully embraced the movement will say there is no reason to add 'Sec' into DevOps, as security is just another ingredient. The DevOps ideal is to break down silos between individual teams (*e.g.,* architecture, development, IT, security, and QA) to better promote teamwork and better incentivize each team member toward the same goals. If security is just another set of skills blended into the overall effort of building and delivering software, there is no reason to call it out any more than quality assurance. Philosophically they're right. But in practice we are not there yet. Developers may embrace the idea, but they generally suck at facilitating team integration. Sure, security is welcome to participate, but it's up to them to learn where they can integrate, and all too often security is asked to contribute skills which they simply do not possess. It's passive-aggressive team building!

We have one more very important reason to use the DevSecOps name: security efforts for code security are very different than efforts to secure infrastructure and supporting components. The security checks to validate application code is secure (DevSec) are very different than the tooling and

processes to verify the supporting infrastructure (SecOps) is secure. These are two different disciplines, with different tooling and approaches, and it is a mistake to discuss one as a subset of the other.

Why Did We Write This Paper?

We discuss the motivation behind our research to help readers understand our goals and what we wish to convey. This is doubly relevant when we *update* a research paper, as it helps us spotlight recent changes in the industry which have made older papers inaccurate or inadequate to describe recent trends. DevOps has matured considerably in four years, so we have a lot to talk about.

This will be a major rewrite of our 2015 research on Building Security into DevOps, with significant additions around common questions security teams ask about DevSecOps and a thorough update on tooling and integration approaches. Much of this paper will reflect 400+ conversations since 2017 across 200+ security teams at Fortune 2000 firms. So we will include considerably more discussion derived from those conversations, and adjust our focus to security teams more than development or IT.

As DevOps has now been around for years, we will forgo a discussion on what DevOps is, why it works. There are plenty of other publications that cover that topic better than we can here. Rather we will focus on the practicalities of how to put together a DevSecOps program.

Different Focus, Different Value

A plethora of new surveys and research papers are available, and some of them are very good. And there are more conferences and online resources popping up than I can count. For example Veracode recently released the latest iteration of its State of Software Security (SoSS) report and it's a monster, with loads of data and observations. Their key takeaways are that the agility and automation employed by DevSecOps teams provide demonstrable security benefits, including faster patching cycles, shorter flaw persistence, faster reduction of technical debt, and 'easier' scanning — which leads to faster problem identification. Sonatype's recently released 2019 State of the Software Supply Chain shows that "Exemplary Project Teams" who leverage DevOps principles drastically reduce code deployment failure rates, and remediate vulnerabilities in half the time of average groups. And we have events like All Day DevOps, where hundreds of DevOps practitioners share stories on cultural transformations, Continuous Integration / Continuous Deployment (CI/CD) techniques, site reliability engineering, and some DevSecOps. All of which is great, and offers qualitative and quantitative data showing why DevOps works and how practitioners are evolving programs.

So **that's not what this paper is about**. Those resources do not address the questions we are asked each and every week.

This paper is about putting together a comprehensive DevSecOps program. Overwhelmingly my questioners ask, "How do I put a DevSecOps program together?" and "How does security fit into DevOps?" They are *not* looking for justification or individual stories about the nuances to address specific impediments. They want a security program in line with peer organizations, which embraces

"security best practices". They need steps one, two and three. These audiences are overwhelmingly comprised of security and IT practitioners, *largely left behind by development teams* who have at least embraced Agile concepts, if not DevOps outright. Their challenge is to understand what development is trying to accomplish, integrate with them in some fashion, and figure out how to leverage automated security testing to be at least as agile as development.

Common Questions

We went through our call notes from the last three years and tallied up the questions we were asked. Following are the most common questions in order of how often they were asked.

- We want to integrate security testing into the development pipeline, and are going to start with static analysis. How do we do this?
- How do we build an application security strategy in light of automation, CI/CD and DevOps?
- How can we start building out an application security strategy? What app security standards should I follow?
- Development is releasing code to production every day. How do we get control over development? Can we realistically modify developer behavior?
- What is the best way to introduce DevSecOps? Where do I start? What are the foundational pieces?
- How do we get different units to adopt the same program (different teams all do things differently)? Right now some use waterfall, some agile, and others DevOps?
- How should we (Security) work with Dev?
- We understand "shift left", but are the tools effective? What tools do you recommend we start with?

These questions contain some common threads: they all come from firms with at least some DevOps teams already in place, where security has some understanding of the intent of DevOps, but they are all starting from scratch. Even security teams with tests already built into their development pipelines struggle with the value each tool provides, pushback from developers over false positives, how to work with developers, or how to scale consistently across many development teams. During calls and engagements, we find security doesn't quite understand why developers embrace DevOps, often missing the thrust of their effort.

The following is a list of questions which security teams should ask but don't.

- How do we fit — operationally and culturally — into DevSecOps?
- How do we get visibility into Development and their practices?

- How do we know changes are effective? What metrics should we collect and monitor?
- How do we support Development?
- Do we need to know how to code?

Bridging The Gap Between Sec and Dev

Automation is key to speed, agility and scalability of the *security team*. Automated security testing, just like automated application build and deployment, takes time and skill to build out. In our typical engagements with clients, *developers* are absent from the calls. A divide still exists, with little communication between security and usually dozens to hundreds of diverse development teams. When developers are present they explain that the security team can create scripts to integrate security testing into the build server, codify security policies, and stitch together security analysis tools with trouble ticketing and metrics with a few lines of python code. After all, many IT practitioners are learning to script for configuration management and build templates to define infrastructure deployments, so why not security? But few security practitioners are capable of coding at this level. Worse, most firms we speak with have a ratio of around 100 developers to every security practitioner, so there is simply no way to scale their security resources across all development projects.

It does not help that many security professionals are early in understanding DevOps, while developers have been adopting various methods over the last decade to become more agile. Security is genuinely behind the curve, and it seems the bulk of the available research, as mentioned above, is not aimed at tackling security's introduction and integration.

This research paper is to help security teams answer the above questions and bridge the gap between them and development.

How Security Works With Development

In our first paper on 'Building Security Into DevOps', given the 'newness' of DevOps for most of our readers, we included a discussion on the foundational principles and how DevOps is meant to help tackle numerous problems common to software delivery. Please refer to that paper if you want more detailed background information. For our purposes here we will discuss just a few principles that directly relate to the integration of security teams and testing with DevOps principles. These concepts lay the foundations for addressing the questions we raised in the first section, and readers will need to understand these as we discuss security tooling and approaches in a DevOps environment.

DevOps and Security

Before we dive in, let's answer one of the most common questions from the previous section: "How do we get control over development?" The short answer is you do not. The longer answer is, in DevOps, you need to work along side your partner, not "control" them. Yes, a small percentage of organizations we spoke with gate all software releases by having security run a battery of tests prior to release, and certify every release from a security standpoint. It is rare that security gets to control software releases in this way, it's anti-DevOps, but it can be very effective security control if not altogether efficient. For that reason, the remainder of this section provides some proven examples of how to work with Development teams, as a partnership, rather that attempt to gain control over them.

Build Security In

It is a terrible truth, but wide use of application security techniques *within the code development process* is relatively new. Sure, the field of study is decades old, but application security was more often bolted on with network or application firewalls, not baked into the code itself. Security product vendors discovered that understanding application requests in order to detect and then block attacks is incredibly difficult to do *outside* the application. It is far more effective to fix vulnerable code and close off attack vectors when possible. Add-on tools are getting better — and some work inside the application context — but better to address the issues in the code when possible.

A central concept when building security in is 'shift left', or the idea that we integrate security testing earlier within the Software Development Lifecycle (SDLC) — the phases of which are typically listed left to right as design, development, testing, pre-production and production. Essentially we shift more resources away from production on the extreme right, and put more into design, testing and

development phases. Born out of lean manufacturing, Kaizen and Deming's principles, these ideas have been proven effective, but typically applied to the manufacture of physical goods. DevOps has promoted use in software development, demonstrating we can improve security at a lower cost by shifting security defect detection earlier in the process.

Automation

Automation is one of the keys to success for most firms we speak with, to the point that the engineering teams often equate DevOps and Automation as synonymous. The reality is the cultural and organizational changes that come with DevOps are equally important, it's just that automation is sometimes the most quantifiable benefit.

Automation brings speed, consistency and efficiency to all parties involved. DevOps, like Agile, is geared towards doing less, better, and faster. Software releases occur more regularly, with less code change between them. Less work means better focus and more clarity of purpose with each release, resulting in fewer mistakes. It also means it's easier to rollback in the event of mistakes. Automation helps people get their jobs done with less hands-on work, but as automation software does exactly the same things every time, consistency is the most conspicuous benefit.

The place where automation is first applied, where the benefits of automation are most pronounced, are the application build servers. Build servers (e.g.: Bamboo, Jenkins, CircleCI), commonly called Continuous Integration (CI) servers, automatically construct an application — and possibly the entire application stack — as code is changed. Once the application is built, these platforms may also launch QA and security tests, kicking back failed builds to the development team. Automation benefits other facets of software production, including reporting, metrics, quality assurance and release management, but security testing benefits are what we are focused on in this research.

On the outset this may not seem like much; calling security testing tools instead of manually running the tests. That perspective misses the fundamental benefits of automated security testing. Automation is how we ensure that *each* update to software includes security tests, ensuring consistency. Automation is how we help avoid mistakes and omissions common with repetitive and — let's be totally transparent here — boring manual tasks. But most importantly, as security teams are typically outnumbered by developers at a ratio of 100 to one, automation is *the* key ingredient to scaling security coverage without having to scale security personnel headcount.

One Team

A key DevOps principle is to break down silos and have better cooperation between developers and supporting QA, IT, security and other teams. We have heard this idea so often that it sounds cliché, but the reality is few in software development actually made changes to implement this idea. Most DevOps-centric firms are changing development team composition to include representatives from all disciplines; that means every team has someone who knows a little security and/or represents security interest, even on small teams. And for those that do, they realize the benefits not just of better communication, but true alignment of goals and incentives. Development in isolation is incentivized to write new features. Quality Assurance in isolation is incented to get code coverage for

various tests. When everyone on a team is responsible for the successful release of new software, there is a change in priorities and changes to behavior.

This item remains a bit of a problem for many of the firms we have interviewed. The majority of firms we have spoken with are large in size, with hundreds of development teams located in different countries, some of which are third-party (re: external) consultants. It is hard to get consistency across all of these teams, and even harder to get general participation. Managerial structure is set up so development managers manage developers, not IT personnel. The management tools for feature tracking, trouble-ticketing, and resource allocation are geared towards a siloed structure. And many leading security tools are set to analyze and report defects to security professionals, not developers or IT personnel who resolve an issue. Progress is still measured in feature outputs and code coverage, and bonuses are awarded accordingly.

The point here is this cultural change, and the great benefits derived are not realized without some changes to the supporting systems and structures. This is a very hard adjustment, one where the various managers are all looking to implement policies as if they have full oversight, missing the point that they too need to adopt the 'one team' approach with their peers to effectively enact changes.

Security Practitioners and Application Security

Why security folks struggle with DevSecOps, and even application security in general, is they do not have backgrounds in software development. Most security practitioners come from a network security background, and many CISOs we speak with are more risk and compliance focused, so there is a general lack of understanding of software development. This lack of knowledge of development tooling and processes, along with common challenges developers are trying to overcome, means security teams seldom understand why automated build servers, central code repositories, containers, Agile and DevOps have caught fire and have been widely adopted in a very short time. Here we discuss some of the drivers for changes in development practices and the key areas security teams need to understand when trying to get a handle on application security.

- **Knowledge of Process**: We are not here to teach the nuances of development process, but we want to point out the reasons why processes change: Speed. For example, moving from Waterfall to various processes like Spiral, Prototype Evolutions, Extreme Programming, Agile and Agile with Scrum and other variations made over the last 20 years. Each was an attempt to satisfy the same goals: Simplify, hopefully avoiding bugs, and speed up delivery. The entirety of software development practice evolution in recent decades has been aimed at addressing these three goals, which helps to show that the process itself is not the important part. Daily scrums, bi-weekly software delivery (sprints), Kanban, Agile, test driven development and automated build servers are *tools* to advance the state of the art. So it is critical for security professionals to understand that security testing and policies should embrace these same ideals. And lastly, DevOps is process independent; you can embrace DevOps and still have a waterfall process, but certainly DevOps fits more naturally with Agile.

- **Knowledge of Tools**: Software development leverages many tools to manage code and process. The two most important to security are code repositories and build tools. Repositories like Git essentially manage application code, giving developers a shared location to store code, track versions and changes. Others, like Docker Registry, are specifically for containers. These tools are essential for developers to manage the code they are building, but also important to security as a place where code can be inspected. Build servers like Jenkins and Bamboo automate the building, testing, and delivery of code. But rather than at a component or module level, they are typically employed for full application stack testing. Developers and quality assurance teams use the build server to launch functional, regression, and unit testing; security teams should leverage these build servers to integrate security testing (such as SAST, DAST, Composition Analysis, and security unit tests) so it fits within the same build process and uses all the same management and communications tools. It is important for security teams to understand which tools the development team uses and who controls those resources, and arrange for integration of security testing.

- **Everything Is Code**: Applications are software. This is fairly well understood, but less appreciated is that in many environments — especially public cloud — servers, networks, messaging, IAM and every other bit of the infrastructure may be defined as configuration scripts, templates, or application code. IT teams now define entire data centers with templates comprised of a few hundred lines of script. The key for security practitioners are twofold: security policies can also be defined in scripts/code, and you can examine code repositories to ensure the templates, scripts and code are secure *before they run*. This is a fundamental change to security audits.

- **Open Source**: Open source software plays a huge part in application development, and is so universally embraced in the development community that it is almost impossible to find a new application development project which does not leverage it. This means a large portion of your code may not be tested the way you expect, or developers may *intentionally* use old, vulnerable versions, because they *know the old version works* with their code. If they change a library, it might break the application and require more work. Developers are incented to get code working and we have witnessed heroic efforts on their part to avoid new (patched) open source versions, for the sake of stability. So we want you to come away with two points: you need to test open source code before it hits production, and you need to ensure that developers do not surreptitiously swap out trusted debugged versions of open source libraries for older, probably vulnerable, versions.

- **Tooling and Developer "Buy in"**: The first step most security teams take when introducing security into application development is to run static analysis scans. The good part is that most security practitioners know what SAST is and does. The bad part is that security started with older SAST tools which were slow, produced output only intelligible by security folks, created "false positive" alerts, and lacked critical APIs needed to fully integrate with the build process. Overall, SAST integration efforts were developer hostile, and most development teams reacted by ignoring the scans or removing those tools from the build process. Two key points: select

tools which fit the development model (faster, easier, better), and use tools which are actually effective. Left to their own decisions, developers always choose the easiest tool to integrate, not the most effective security scanner. It is important that the security team be part of the security tool selection process to ensure security scans provide adequate analysis.

- **Security Friction & Cultural Dynamics**: Most application security teams are playing catch-up. Development is (usually) already agile, and if some of your supporting organizations are embracing DevOps, IT and QA may be as well. This means security folks are the non-agile anomaly because *anything* you do or ask adds time and complexity, the antithesis of software engineering goals. This topic is so important that I added the entire next section, "Scaling Security", to address the cultural friction between security and development.

- **SDLC and S-SDLC**: Many application security teams approach application security by looking at the Software Development LifeCycle (SDLC), with the goal of applying some form of security analysis in each phase of the lifecycle. A Secure SDLC (S-SDLC) typically includes threat modeling during design, composition analysis in development, static analysis during build, and any number of tests pre-production. This is an excellent way to set up a *process independent* application security program. As many large organizations come to understand, each of your development teams employs a slightly different process, and it is entirely possible your company uses every known development process in existence. This is a huge headache but the S-SDLC can become your yardstick: Use it as your policy template and map security controls to the fit within the different processes.

Scaling Security

As mentioned in the introductory section, security teams are vastly outnumbered. As an example, I spoke with three midsized firms this week — their development personnel ranged from 800-2000 people, while their security teams ranged from 12 to 25. They typically had two or three security personnel with backgrounds in application security. They may be rare as unicorns, but that does not give them magical powers to cover all development operations, so they need ways to scale their experience across the enterprise. And they need to do it in a way which meshes with development objectives, getting software development teams to implement their security controls. Here are several methods that work.

- **Automation**: We have already discussed automation to some degree so I can keep it short here. Automation is how security analysis can be faster, more frequent, and without direct involvement from the security team. Security tools which perform automated analysis, either out of the box or your own custom checks, are critical to scale across development teams. And yes, this means you need to integrate security tools into every build pipeline in your company. But this means that not only are scans automated, but distribution of results is also integrated with other tools and processes. This is how teams scale, and necessary for the next two items.

- **Failing the Build**: Development and security teams commonly have friction with each other. Security typically provides development managers security scan results with thousands of

defects. Development managers read them as saying "Dude, your code sucks, what's wrong with you, fix it now!" One way to reduce friction between the two groups is to take the output from static or dynamic scans, discuss the scope of the problem, what critical defects mean, and come to agreement on what is reasonable to fix in the middle term. Once everyone agrees about what a critical issue is and what is a reasonable timeframe to fix it, you instruct the security tools to fail a build when critical bugs are discovered. This process takes some time to implement, and some pain to work through, but it changes the relationship between security and development. No longer is it security saying code is defective — instead it's an unbiased tool reporting a defect to development. Security is no longer the bad guy standing in the way of progress— now development must meet a new quality standard, one focused on code quality in terms of security defects. It also shifts interactions because developers often need assistance understanding the defect, look for ways to tackle classes of defects instead of individual bugs, and seek help from security. Failing the build creates a sea change between groups. Any security tooling which produces false positives magnifies the difficulty in establishing this change, but this step is critical for DevOps teams.

- **Metrics**: Metrics are essential to understand the scope of application security issues, and security tools are how you collect most metrics. Even of you do not fail the build, and even if the results are not shared with anyone outside security, integrating security testing into build servers and code repositories is critical to gaining visibility and producing metrics. These metrics help you decide where to spend budget — whether on additional tooling, developer education, or runtime protection. And these metrics will be your guide to the effectiveness of the tools, education, and runtime protection you implement. Without metrics you're just guessing.

- **Security Champions**: One of the most effective methods I have discovered to scale security is to deputize willing developers with an active interest in security to be "security champions" on their development teams. Most developers have some interest in security, and they know security education makes them more valuable, which often means raises. For questions on security the developer gains a liaison on a security team, and in turn can ask the champion their own questions. Typically security teams cultivate these relationships through education, a "center of excellence" where developers and security pros can interact (often a Slack channel these days), sending developers to security conferences, or simply sponsoring events like lunches where security topics are discussed. Regardless of how you do it, this is an excellent way to scale security without scaling headcount, and we recommend you set aside some budget and resources — it returns far more benefits than it costs.

- **Education**: If you want developers to understand security and threats to applications, educate them. Educational budgets for engineering leads and VPs are usually tightly restricted. To fill gaps it is not uncommon for security teams to shoulder the expense of educating select developers on relevant skills the organization lacks. Sometimes this is through purchase of security related CBT, sometimes through purchase of professional services from security vendors, and sometimes it is specific classes from SANS or other organizations. Understanding

how to remediate application security issues, security reference architectures, how to perform threat modeling, and how to use security tools are all common topics.

Security Planning

This section is intended to help security folks create an outline or *structure* for an application security program. We are going to answer such common questions as "How do we start building out an application security strategy?", "How do I start incorporating DevSecOps?" and "What application security standards should I follow?". I will discuss the Software Development Lifecycle (SDLC), introduce security items to consider as you put your plan in place, and reference some application security standards for use as guideposts for what to protect against. This section will help your strategy; the next one will cover tactical tool selection.

Security Planning and your SDLC

A Secure Software Development Lifecycle (S-SDLC) essentially describes how security fits into the different phases of a Software Development Lifecycle. We will look at each phase in an SDLC and discuss which security tools and techniques are appropriate. Note that an S-SDLC is typically drawn as a waterfall development process, with different phases in a linear progression, but that's really just for clearer depiction — the actual SDLC which is being secured is as likely to be Agile, Extreme, or Spiral as Waterfall. There are good reasons to base an S-SDLC on a more modern SDLC; but the architecture, design, development, testing, and deployment phases all map well to development stages in any development process. They provide a good jumping-off point to adapt current models and processes into a DevOps framework.

As in our previous section, we want you to think of the S-SDLC as a framework for building your security program, not a full step-by-step process. We recognize this is a departure from what is taught in classrooms and wikis, but it is better for planning security in each phase.

Define and Architect

- **Reference Security Architectures**: Reference security architectures exist for different types of applications and services, including web applications, data processing applications, identity and access management services for applications, stream/event processing, messaging, and so on. The architectures are even more effective in public cloud environments, Kubernetes clusters, and service mesh environments — where we can tightly control via policy how each application operates and communicates. With cloud services we recommend you leverage service provider guidelines on deployment security, and while they may not call them 'reference security architectures' they do offer them. Educate yourself on the application platforms and ask software designers and architects which methods they employ. Do not be surprised if for legacy applications they give you a blank stare. But new applications should include plans for process isolation, segregation, and data security, with a full IAM model to promote segregation of duties and data access control. Operational Standards: Work with your development teams to define

minimal security testing requirements, and critical and high priority issues. You will need to negotiate which security flaws will fail a build, and define the process in advance. You will probably need an agreement on timeframes for fixing issues, and some type of virtual patching to address hard-to-fix application security issues. You need to define these things up front and make sure your development and IT partners agree.

- **Security Requirements**: Just as with minimum functional tests which must run prior to code acceptance, you'll have a set of security tests you run prior to deployment. These may be an agreed upon battery of unit tests for specific threats which your team writes. Or you may require all OWASP Top Ten vulnerabilities be mitigated in code or supporting products, mapping each threat to a specific security control for all web applications. Regardless of what you choose, your baseline requirements should account for new functionality as well as old. A growing body of tests requires more resources for validation and can slow your test and deployment cycle over time, so you have some decisions to make regarding which tests can block a release vs. what you scan for post-production.

- **Monitoring and Metrics**: If you will make small iterative improvements with each release, what needs fixing? Which code modules are problematic for security? What is working and how can you prove it? Metrics are key to answering all these questions. You need to think about what data you want to collect and build it into your CI:CD and production environments to measure how your scripts and tests perform. That means you need to engage developers and IT personnel in collecting data. You'll continually evolve collection and use of metrics, but plan for basic collection and dissemination of data from the get-go.

Design

- **Security Design Principles**: Some application security design and operational principles offer significant security improvement. Things like ephemeral instances to aid patching and reduce attacker persistence, immutable services to remove attack surface, configuration management to ensure servers and applications are properly set up, templating environments for consistent cloud deployment, automated patching, segregation of duties by locking development and QA personnel out of production resources, and so on. Just as important, these approaches are key to DevOps because they make delivery and management of software faster and easier. It sounds like a lot to tackle, but IT and development typically pitch in as it makes their lives easier too.

- **Secure the Deployment Pipeline**: With production environments more locked down, development and test servers become more attractive targets. Traditionally these environments run with little or no security. But the need for secure source code management, build servers, and deployment pipelines is growing. And as CI/CD pipelines offer an automated pathway into production, you'll need at minimum stricter access controls for these systems — particularly build servers and code repositories. And given scripts running continuously in the background with minimal human oversight, you'll need additional monitoring to catch errors and misuse. Many of the tools offer good security, with digital fingerprinting, 2FA, logging, role-based access

control, and other security features. When deployed in cloud environments, where the management plane allows control of your entire environment, great care must be taken with access controls and segregation of duties.

- **Threat Modeling**: Threat modeling remains one of the most productive exercises in security. DevOps does not change that, but it does open up opportunities for security team members to instruct dev team members on common threat types, and to help plan out unit tests to address attacks. This is when you need to decide whether you will develop this talent in-house or engage a consultant as there really is no product to do this for you. Threat modeling is often performed during design phases, but can also occur as smaller units of code are developed, and sometimes enforced with home-built unit tests.

Develop

Infrastructure and Automation First: Automation and Continuous Improvement are key DevOps principles, and just as valuable for security. And automation is essential, so you need to select and deploy security tooling. We stress this because planning both how and who will deploy the tools as this is likely beyond the skills of the average security team member. And it helps development plan out the tools and tests they need to deploy before they can deliver new code. Keep in mind that many security tools require some development skill to integrate, so plan either to get your staff to help, or engage professional services. The bad news is that there is up-front cost and work to be done in preparation; the good news is that each and every build in the future will benefit from these efforts.

- **Automation First**: Remember that development is not the only group writing code and building scripts — operations is now up to their elbows as well. This is how DevOps helps bring patching and hardening to a new level. Operations' DevOps role is to provide build scripts which build out the infrastructure for development, testing, and production servers. The good news is that you are now testing exact copies of production. Templates and configuration management address a problem traditional IT has struggled with for years: *ad hoc* undocumented work that 'tweaks' the environment to get it working. Again, there is a great deal of work to get environments fully automated — on servers, network configuration, applications, and so on — but it makes future efforts faster and more consistent. Most teams we spoke with build new machine images every week, and update their scripts to apply patches, updating configurations and build scripts for different environments. But this work ensures consistency and a secure baseline.

- **Secure Code Repositories**: You want to provide developers an easy way to get secure and (internally) approved open source libraries. Many clients of ours keep local copies of approved libraries and make it easy to get access to these resources. Then they use a combination of composition analysis tools and scripts, before code is deployed into production, to ensure developers are using approved versions. This helps reduce use of vulnerable open source.

- **Security in the Scrum**: As mentioned in the previous section, DevOps is process neutral. You can use Spiral, or Agile, or surgical-team approach as you prefer. But Agile Scrums and Kanban techniques are well suited to DevOps. Their focus on smaller, focused, quickly demonstrable tasks aligns nicely. We recommend setting up your "security champions" program at this time, training at least one person on each team on security basics, and determining which team members are interested in security topics. This way security tasks can easily be distributed to team members with interest and skill in tackling them.

- **Test Driven Development**: A core tenet of Continuous Integration is to never check in broken or untested code. The definitions of broken and untested are up to you. Rather than writing giant waterfall-style specification documents for code quality or security, you're documenting policies in functional scripts and programs. Unit tests and functional tests not only define but *enforce* security requirements. Many development teams use what is called "test driven development", where the tests to ensure desired functionality — and avoid undesired outcomes — are constructed along with the code. These tests are checked in and become a permanent part of the application test suite. Security teams do not leverage this type of testing enough, but this is an excellent way to detect security issues specific to code which commercial tools do not.

- **IAST**: Interactive Application Security Testing is where you analyze your application's code, usually with a scanning tool designed for the purpose, for security vulnerabilities before you launch code into production. The idea is to test code earlier in the process, but particular to IAST is the ability to 'interactively' run security tests against new code. These tools often rely upon SAST and DAST like components to instrument code and then crawl the application while performing tests. There are several deployment models for IAST including desktop IDE plug-ins, tools run prior to source code commits, and embedded into QA tools. IAST is not a new idea, but the market is not mature.

Test

- **Design for Failure**: DevOps turns many long-held principles of both IT and software development upside down. For example durability used to mean 'uptime', but now it's speed of replacement. Huge documents with detailed product specifications have been replaced by Post-It notes. And for security, teams once focused on getting code to pass functional requirements now look for ways to break applications before someone else can. This new approach of "chaos engineering", which intentionally breaks application deployments, forces engineers to build in reliability *and security*. A line from James Wickett's GauntIt page: Be Mean To Your Code — And Like It expresses the idea eloquently. The goal is not just to test functions during automated delivery, but to really test the ruggedness of code, and substantially raise the minimum security of an acceptable release. We harden an application by intentionally pummeling it with all sorts of functional, stress, and security tests before it goes live — reducing the time required for security experts to test code hands-on. If you can figure out some way to break your application, odds are attackers can too, so build the test — and the remedy — *before* it goes live. You need to plan for these tests, and the resources needed to build them.

- **Parallelize Security Testing**: A problem common to all Agile development approaches is what to do about tests which take longer than a development cycle. For example we know that fuzz testing critical pieces of code takes longer than an average Agile sprint. SAST scans of large bodies of code often take an order of magnitude longer than the build process. DevOps is no different — with CI and CD code may be delivered to users within hours of its creation, and it may not be possible to perform complete white-box testing or dynamic code scanning. To address this issue DevOps teams run multiple security tests in parallel to avoid delays. They break down large applications into services to speed up scans as well. Validation against known critical issues is handled by unit tests for quick spot checks, with failures kicking code back to the development team. Code scanners are typically run in parallel with unit or other functional tests. Our point here is that you, as a security professional, should look for ways to speed up security testing. Organizing tests for efficiency vs. speed — and completeness vs. time to completion — was an ongoing balancing act for every development team we spoke with. Focusing scans on specific areas of code helps find issues faster. Several firms also discussed plans to maintain pre-populated and fully configured tests servers — just as they do with production servers — waiting for the next test cycle to avoid latency. Rewriting and reconfiguring test environments for efficiency and quick deployments help with CI.

Pre-Release

- **Elasticity FTW**: With the public cloud and virtualized resources it has become much easier to quickly provision test servers. We now have the ability to spin up new environments with a few API calls and shrink them back down when not in use. Take advantage of on-demand elastic cloud services to speed up security testing.

- **Test Data Management**: Developers and testers have a very bad habit of copying production data into development and test environments to improve their tests. This had been the source of many data breaches over the last couple decades. Locking down production environments so QA and Dev personnel cannot exfiltrate regulated data is great, but also ensure they do not bypass your security controls. Data masking, tokenization, and various tools can produce quality test data, minimizing their motivation to use production data. These tools deliver test data derived from production data, but stripped of sensitive information. This approach has proven successful for many firms, and most vendors offer suitable API or automation capabilities for DevOps pipelines.

Deploy

- **Manual vs. Automated Deployment**: It is easy enough to push new code into production with automation. Vetting that code, or rolling back in case of errors, is much harder. Most teams we spoke with are not yet completely comfortable with fully automated deployment — it scares the hell out of many security folks. Continuous software delivery is really only used by a small

minority of firms. Most only release new code to customers every few weeks, often after a series of sprints. These companies execute many deployment actions through scripts, but launch the scripts manually when Operations and Development resources are available to fully monitor the push. Some organizations really are comfortable with fully-automated pushes to production, releasing several times per day. There is no single right answer, but either way automation performs the bulk of the work, freeing up personnel to test and monitor.

- **Deployment and Rollback**: To double-check that code which worked in pre-deployment tests still works in the development environment, teams we spoke with still do 'smoke' tests, but they have evolved them to incorporate automation and more granular control over rollouts. We saw three tricks commonly used to augment deployment. The first and most powerful is called Blue-Green or Red-Black deployment. Old and new code run side by side, each on its own set of servers. A rollout is a simple flip at the load balancer level, and if errors are discovered the load balancers are pointed back to the older code. The second, canary testing, is where a small subset of individual sessions are directed towards the new code — first employee testers, then a subset of real customers. If the canary dies (errors are encountered), the new code is retired until the issued can be fixed, when the process is repeated. Finally, feature tagging enables and disables new code elements through configuration files. If event errors are discovered in a new section of code, the feature can be toggled off until it is fixed. The degrees of automation and human intervention vary greatly between models and organizations, but overall these deployments are far more automated that traditional web services environments.

- **Production Security Tests**: Applications often continue to function even when security controls fail. For example a new deployment scripts might miss an update to web application firewall policies, or an application could launch without any firewall protection. Validation — at least sanity checks on critical security components — is essential for the production environment. Most of the larger firms we spoke with employ penetration testers, and many have full-time "Red Teams" examining application runtime security for flaws.

- **Automated Runtime Security**: Many firms employ Web Application Firewalls (WAF) as part of their application security programs, usually in order to satisfy PCI-DSS requirements. Most firms we spoke with were dissatisfied with these tools, so while they continue to leverage WAF blacklists, they were adopting Runtime Application Self-Protection (RASP) to fill remaining gaps. RASP is an application security technology which embeds into an application or application runtime environment, examining requests at the application layer to detect attacks and misuse in real time. Beyond just "WAF in the application context", RASP can monitor and enforce at many points within an application framework, both tailoring protection to specific types of attacks and allowing web application requests to "play out" until it becomes clear a request is indeed malicious before blocking it. Almost every application security and DevOps call we took over the last three years included discussion of RASP, and most firms we spoke with have deployed the technology.

Application Security Standards

A handful of application security standards are available. The Open Web Application Security Project (OWASP) Top Ten and the SANS Common Weakness Enumeration Top 25 are the most popular, but other lists of threats and common weaknesses are available, typically focused on specific subtopics such as CSA's top threats to cloud computing or application security measurement. Each tends to be embraced by one or more standards organizations, so which you use is generally dictated by which industry you are in. Or you can use all of them.

Regardless of your choice, the idea is to understand what attacks are common and account for them with one or more security controls and application security tests in your build pipeline. Essentially you build out a matrix of threats, and map them to security controls. This step helps you plan out what security tools you will adopt and put into your build process, and which you will use in production.

Security Tools and Testing

In this section we show you how to weave security into the fabric of your DevOps automation framework. We are going to address the questions "We want to integrate security testing into the development pipeline, and are going to start with static analysis. How do we do this?", "We understand “shift left”, but are the tools effective?" and "What tools do you recommend we start with, and how do we integrate them?". As DevOps encourages testing in all phases of development and deployment, we will discuss what a build pipeline looks like, and the tooling appropriate for each stage. The security tests typically sit side by side with functional and regression tests your quality assurance teams has likely already deployed. And beyond those typical post-build testing points, you can include testing on a developer's desktop prior to check-in, in the code repositories before and after builds, and in pre-deployment staging areas.

Build Process

During a few of the calls we had, several of the senior security executives did not know what constituted a build process. This is not a condemnation as many people in security have not participated in software production and delivery, so we want to outline the process and the terminology used by developers. If you're already familiar with this, skip forward to 'Building a Security Toolchain'.

Most of you reading this will be familiar with "nightly builds", where all code checked in the previous day is compiled overnight. And you're just as familiar with the morning ritual of sipping coffee while you read through the logs to see if the build failed and why. Most development teams have been doing this for a decade or more. The automated build is the first of many steps that companies go through on their way toward full automation of the processes that support code development. Over the last several years we have mashed our 'foot to the floor', leveraging more and more automation to accelerate the pace of software delivery.

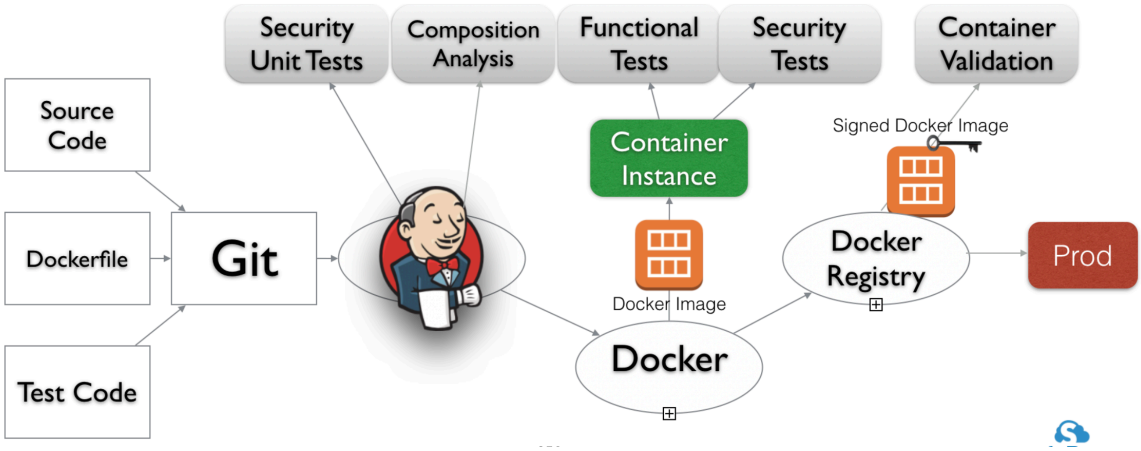
The path to DevOps is typically taken in two phases: first with continuous integration, which manages the building and testing of code; then continuous deployment, which assembles the entire application stack into an executable environment. And at the same time, there are continuous improvements to all facets of the process, making it easier, faster and more reliable. It takes a lot of work to get here, and the scripts and templates used often take months just to build out the basics, and years to mature them into reliable software delivery infrastructure.

Continuous Integration

The essence of Continuous Integration (CI) is developers regularly check-in small iterative code advances. For most teams this involves many updates to a shared source code repository, and one or more builds each day. The key is smaller, simpler additions, where we can more easily and quickly find code defects. These are essentially Agile concepts, implemented in processes which drive code, rather than processes that drive people (such as scrums and sprints). The definition of CI has evolved over the last decade, but in a DevOps context CI implies that code is not only built and integrated with supporting libraries, but also automatically dispatched for testing. Additionally, DevOps CI implies that code modifications are not applied to branches, but directly into the main body of the code, reducing the complexity and integration nightmares that can plague development teams.

This sounds simple, but in practice it requires considerable supporting infrastructure. Builds must be fully scripted, and the build process occurs as code changes are made. With each successful build the application stack is bundled and passed along for testing. Test code is built before unit, functional, regression, and security testing, and checked into repositories and part of the automated process. Tests commence automatically whenever a new application bundle is available, but it means the new tests are applied with each new build as well. It also means that before tests can be launched test systems must be automatically provisioned, configured, and seeded with the necessary data. Automation scripts must provide monitoring for each part of the process, and communication of success or failure back to Development and Operations teams as events occur. The creation of the scripts and tools to make all this possible requires Operations, Testing and Development teams to work closely together.

The following graphic shows an automated build pipeline, including security test points, for containers. Again, this level of orchestration does not happen overnight, rather an evolutionary process that take months to establish the basics and years to mature. But that is the very essence of continuous improvement.



Continuous Deployment

Continuous Deployment looks very similar to CI, but focuses on the *releasing software to end users* rather than building it. It involves similar packaging, testing, and monitoring tasks, with some additional wrinkles. Upon a successful completion of a build cycle, the results feed the Continuous Deployment (CD) process. CD takes another giant step forward for automation and resiliency but automating the release management, provisioning and final configuration for the application stack, then launches the new application code.

When we talk about CD, there are two ways people embrace this concept. Some teams simply launch the new version of their application into an existing production environment. The CD process automates the application layer, but not the server, data or network layers. We find this common with on-premise applications and in private cloud deployments, and some public cloud deployments still use this model as well.

A large percentage of the security teams we spoke with are genuinely scared of Continuous Deployment. They state "How can you possibly allow code to go live without checks and oversight!", missing the point that the code does not go live until all of the security tests have been passed. And some CI pipelines contain manual inspection points for some tests. In our experience, CD means *more reliable and more secure* releases. CD addresses dozens of issues that plague code deployment — particularly in the areas of error-prone manual changes, and discrepancies in revisions of supporting libraries between production and development. Once sufficient testing is in place, there should be no reason to mistrust CD.

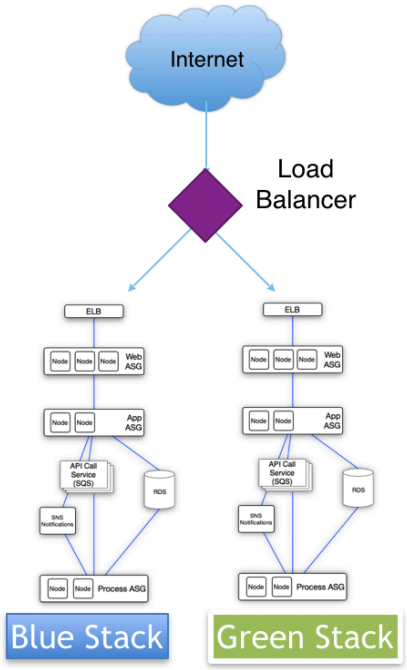
Not all firms release code into production every day; in fact less than 10% of the firms we speak with truly release continually, but some notable companies like Netflix, Google and Etsy have automated releases once their tests have completed. But most firms (*i.e.*: those not in the content or retail verticals) do not have a good business need to release updates multiple times a day, so they don't.

Managed / Blue-Green Deployments

Most firms have a slower release cycle, often with a 'go live' cadence of every one to three sprints. We call these 'managed releases' as the execution and timing is manual, but most of the actions are automated. Plus these firms employ another very powerful technique: Automated infrastructure deployments. This is where you cycle the entire infrastructure stack along with the application. These types of deployments rely upon automating the environment the software runs in; this can be as simple as standing up a Kubernetes cluster, or leveraging Openshift to run Terraform templates into Google GCP, or launching an entire AWS environment via Cloudformation templates. The infrastructure and the application are all code, and so both are launched in tandem. This is becoming common in public cloud deployments.

But this method of release offers significant advantages and provides the foundation from what is called 'Blue-Green' or Red-Black deployment. Old and new code run side by side, looking close to mirror images, each on their own set of servers. While the old code (i.e.: Blue) continues to serve

user requests, new code (i.e.: Green) is exercised only by select users or test harnesses. A rollout is a simple redirection at the load balancer level, and internal users and live customers can be slowly re-directed to the Green servers, essentially using them as testers for the new system. If the new code passes all required tests, the load balancer sends all traffic to the Green servers, the Blue servers are retired, and Green then becomes the new Blue. If errors are discovered, the load balancers are pointed back to the old 'Blue' code until a new patched version is available. This is essentially pre-release testing in production, and near instantaneous rollback in the even there are defects or security problems discovered.



Where to Test

Desktop Security Tests

Integrated Development Environments, or IDEs, are the norm for most developers. Visual Studio, Eclipse, IntelliJ and many more, some tailored to specific languages or environments. These desktop tools not only help with building code, but they integrate syntax checkers, runtimes, terminals, packages, and lots of other features to make building code easier. Commercial security vendors have plug-ins for the popular tools, usually providing a form of static analysis. Sometimes these tools are interactive, giving advice as code is written, while others check currently modified files on demand, before code is committed. There are even one or two that do not actually 'plug in' to the IDE, but work as a standalone tool and run prior to checkin. As the code scans are typically just the module or container that the developer is working on, code scans happen very quickly. And getting the security scans right at this stage lowers the likelihood that the build server will find and fail on security issues after code is checked in.

The development teams we have worked with that use these tools find them to be effective and deliver as promised. Many individual developers we speak with do not like many of these security plug-ins as they find them noisy and distracting. We recommend the use of these desktop scanners when possible but recognize the cultural impediment to use, and caution that security teams may need to help change the culture and allow adoption to grow over time.

Code Repository Scanning

Source code management, configuration management databases, container registries and similar types of tools store code and help with management tasks like versioning, IAM and approval processes. From a security standpoint, several composition analysis vendors have integrated their products to check that open source versioning is correct and that the platforms in use do not

contain known CVEs. Additional facilities to create digital fingerprints for known good version and other version control features are becoming more common.

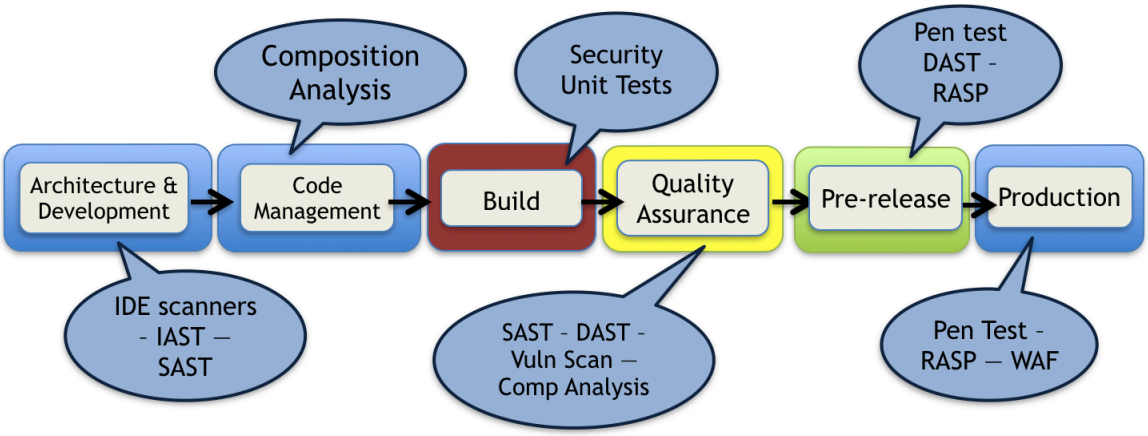
Build Server Integration

Build servers build applications. Often comprised of multiple sources of in-house developed and open source code, it is common for many 'artifacts' to be used in the construction of an application. Build servers like Jenkins and Bamboo fortunately have the hooks needed to massage these artifacts, both before and after a build. This is commonly how testing is integrated into the build pipeline. Leveraging this capability will be central to your security test integration. Composition analysis, SAST and custom tests are commonly integrated at this stage.

Pre-release

For 'code complete' or systemic testing, where all of the parts of the application and the supporting application stack are assembled, a pre-production 'staging area' is setup to mimic the production environment and facilitate a full battery of tests. We are finding several trends of late, and pre-production testing is one of them. As public cloud resources allow for rapid elasticity and on-demand resource procurement, firms are spinning up test environments, running their QA and security tests, then shutting them back down again to reduce costs. In some cases this is used to do DAST testing that used to be performed in production. And in most cases this is the means for Blue-Green deployment model is leveraged to run many different types of testing on anew environment parallel to the existing production environment.

Building a Security Toolchain



Static Analysis

Static Application Security Testing (SAST) examines all code — or runtime binaries — to support a thorough search for common vulnerabilities. These tools are highly effective at finding flaws, even in code that has been manually reviewed. Your selection criteria will likely boil down to speed of scan, ease of integrations, readability of the results and lack of false positives. Most of these platforms have gotten much better at providing analysis that is useful for developers, not just security geeks.

And many of the products are being updated to offer full functionality via APIs or build scripts. If you have a choice, select tools with APIs for integration into the DevOps process, and which don't require "code complete". We have seen a slight reduction in use of these tests, as they often take hours or days to run — in a DevOps environment that can prevent them from running inline as a gate to certification or deployment. As we mentioned in the above under 'Other', most teams are adjusting to support out-of-band — or what we are calling ‘Parallelized’ — testing for static analysis. We highly recommend keeping SAST testing inline if possible, and focus on new sections of code to reduce runtime.

Dynamic Analysis

Rather than scanning code or binaries like SAST, Dynamic Application Security Testing (DAST) dynamically 'crawls' through an application's interface, testing how it reacts to various inputs. These scanners cannot see what's going on behind the scenes, but they offer valuable insight into how code behaves, and can flush out errors which other tests may not see in dynamic code paths. The good news is they tend to have low rates of false positives. These tests are typically run against fully built applications, and can be destructive, so the tools often offer settings to run more aggressively in test environments. And like SAST may require some time to fully scan code, so in line tests that gate a release are often run against new code only, and full application sweeps are run ‘in parallel’.

Composition and Vulnerability Analysis

Composition Analysis tools check versions of open source libraries to assess open source risk, both with security vulnerabilities and potential licensing issues. Things like Heartbleed, misconfigured databases, and Struts vulnerabilities may not be part of your application testing at all, but they all critical application stack vulnerabilities. Some people equate vulnerability testing with DAST, but there are other ways to identify vulnerabilities. In fact there are several kinds of vulnerability scans; some look settings like platform configuration, patch levels or application composition to detect known vulnerabilities. Make sure you broaden you scans to include your application, your application stack, and the open source platforms that support it.

Manual Code Review

Some organizations find it more than a bit scary to fully automate deployment, so they want a human to review changes before new code goes live — we understand. But there are very good security reasons for review as well. In an environment as automation-centric as DevOps, it may seem antithetical to use or endorse manual code reviews or security inspection, but manual review is still highly desirable. Some types of vulnerabilities are not part of scanning tools. Manual reviews often catch obvious stuff that tests miss, and developers can miss on their first (only) pass. And developers' ability to write security unit tests varies. Whether through developer error or reviewer skill, people writing tests miss stuff which manual inspections catch. Your tool belt should include manual code inspection — at least periodic spot checks of new code or things like Dockerfile which are often omitted from scans.

Security Unit Tests

Unit testing is where you check small sub-components or fragments ('units') of an application. These tests are written by programmers as they develop new functions, and commonly run by developers prior to code check-in, but possibly within the build pipeline as well. These tests are intended to be long-lived, checked into the source repository along with new code, and run by every subsequent developers who contributes to that code module. For security these may straightforward — such as SQL injection against a web form — or more sophisticated attacks specific to the function under test, such as business logic attacks — all to ensure that each new bit of code correctly reflects the developers' intent. Every unit test focuses on specific pieces of code — not systems or transactions. Unit tests attempt to catch errors very early in the process, per Deming's assertion that the earlier flaws are identified, the less expensive they are to fix. In building out unit tests you will need to support developer infrastructure to embody your tests, and also to encourage the team to take testing seriously enough to build good tests. Having multiple team member contributes to the same code, each writing unit tests, helps identify weaknesses a single programmer might not consider.

Security Regression Tests

A regression test verifies that recently changed code still functions as intended. In a security context this is particularly important to ensure that vulnerabilities remain fixed. DevOps regression tests are commonly run parallel to functional tests — after the code stack is built out. And in some cases this may need to be in a dedicated environment, where security testing can be destructive and cause side effects that are unacceptable in production servers with real customer data. Virtualization and cloud infrastructure are leveraged to expedite instantiation of new test environments. The tests themselves are typically home-built test cases to exploit previously discovered vulnerabilities, either as unit or systemic tests. The author uses these types of tests to ensure that credentials like passwords and certificates are not included in files, and that infrastructure does not allow port 22 or 3389 access.

Chaos Engineering

Chaos engineering is where random failures are introduced, usually in the production environment, to see how application environments handle adverse conditions. Firms like Netflix has pioneered efforts in this field in order to force their development teams to understand common failure types, and build graceful failure and recovery into their code. From a security standpoint, if an attacker can force an application into a bad state, they can often coerce the application to perform tasks it was not intended to take. Building ruggedness into the code improves reliability and security.

Fuzzing

At its simplest fuzz testing is essentially throwing lots of random garbage at applications, seeing whether any particular (type of) garbage causes errors. Many dynamic scanning vendors will tell you they provide fuzzing. They don't. Go to any security conference — Black Hat, DefCon, RSA, or B-Sides — and you will see that most security researchers prefer fuzzing to find vulnerable code. It has become essential for identifying misbehaving code which may be exploitable. Over the last 10 years, with Agile development processes and even more with DevOps, we have a steady decline in use of fuzz testing by development and QA teams. This is because it's slow; running through a large test

body of possible malicious inputs takes substantial time. This is a little less of an issue with web applications because attackers can't throw everything at the code, but much more problematic for applications delivered to users (including mobile apps, desktop applications, and automotive systems). We almost excluded this section as it is rare to see true fuzz testing in use, but for critical systems, periodic — and out of band — fuzz testing should be part of your security testing efforts.

Risk and Exposure Analysis

Integrating security findings from application scans into bug tracking systems is not that difficult technically. Most products offer it as a built-in feature. The hard part is figuring out what to do with the data once obtained. Is a discovered security vulnerability a real issue? If it is not a false positive, can the vulnerability be exploited? What is its criticality and priority, relative to everything else? And if we choose to address it, how should we deal with it from a set of options (unit test, patch, RASP)?

Another aspect to consider is this information distributed without overloading stakeholders. With DevOps you need to close the loop on issues within infrastructure, security testing as well as code. And Dev and Ops offer different possible solutions to most vulnerabilities, so the people managing security need to include operations teams as well. Patching, code changes, blocking, and functional whitelisting are all options for closing security gaps; so you'll need both Dev and Ops to weigh the tradeoffs.

Security's Role In DevOps

As we mentioned earlier, DevOps is not all about tools and technology — much of its success lies in how people work within the model. We have already gone into great detail about tools and process, and we approached much of this content from the perspective of security practitioners getting onboard with DevOps. This paper is geared toward helping security folks, so here we outline their role in a DevOps environment. We hope to help you work with other teams and reduce friction.

And while we deliberately called this paper "Enterprise DevSecOps", please keep in mind that your development and IT counterparts likely think there is no such thing. To them security becomes part of the operational process of integrating and delivering code. We call security out as a separate thing because, even woven into the DevOps framework, it's substantially more difficult for security personnel to fit in. You need to consider how you can improve delivery of secure code without waste and without introducing bottlenecks in a development process you may not be intimately familiar with. The good news is that security fits nicely within a DevOps model, but you need to tailor things to work within your organization's automation and orchestration to be successful.

The Security Pro's Responsibilities

- **Learn the DevOps model**: We have not even touched on theory or practice of DevOps in this series. There is a lot for you to learn on base concepts and practices. To work in a DevOps environment you need to understand what it is and how it works. You need to understand cultural and philosophical changes as well as how they affect process, tooling, and priorities. You need to understand your organization's approach to optimally integrate security tooling and metrics. Once you understand the mechanics of the development team, you'll have a better idea of how to work with them in their process.

- **Learn how to be agile**: Your participation in a DevOps team means you need to fit into DevOps — not the other way around. The goal of DevOps is fast, faster, fastest: small iterative changes which offer quick feedback, ultimately reducing Work In Progress (WIP). Small, iterative changes to improve security fit this model. Prioritize things which accelerate delivery of secure software over delivery of new features — keeping in mind that it is a huge undertaking to change culture away from "feature first". You need to adjust requirements and recommendations so they fit into the process, often simplified into small steps, with enough information for tasks to be both automated and monitored. You can recommend manual code reviews or fuzz testing, so long as you understand where they fit within the process, and what can — and cannot — gate releases.

- **Educate**: Our experience shows that one of the best ways to bring a development team up to speed in security is training: in-house explanations or demonstrations, third-party experts to help with application security or threat modeling, eLearning, or various commercial courses. The historical downside has been cost, with many classes costing thousands of dollars. You'll need to evaluate how best to use your resources — the answer typically includes some eLearning for all employees, and select people attending classes and then teaching peers. On-site experts can be expensive, but an entire group can participate in training.

- **Grow your own support**: There is simply no way for security teams to scale out their knowledge without help. This does not mean hundreds of new security employees — it means deputizing developers to help with product security. Security teams are typically small and outnumbered by developers 100:1. What's more, security people are not present at most development meetings, so they lack visibility into day-to-day agile activities. To help extend the reach of the security team, see if you can get someone on each development team — a "security champion" — to act as a security advocate. This helps not only extend the security team's reach, but also expand security awareness.

- **Help DevOps team understand threats**: Most developers don't fully grasp how attackers approach systems, or what it means when a code or SQL injection attack is possible. The depth and breadth of security threats is outside their experience and most firms do not teach threat modeling. The OWASP Top Ten is a good guide to the types of code deficiencies which plague development teams, but you should map these threats back to real-world examples, show the damage that a SQL injection attack can cause, and explain how a Heartbleed type vulnerability can completely expose customer credentials. You can these real-world use cases as "shock and awe" to help developers "get it".

- **Advise on remediation practices**: Your security program is inadequate if it simply says to "encrypt data" or "install WAF". All too often, developers and IT have a singular idea of what constitutes security, centered on a single tool they want to set and forget. Help build out the elements of the security program, including both in-code enhancements and supporting tools. Teach how those each help to address specific threats, and offer help with deployment and policy setup. In the past, firms used to produce code style guides to teach younger developers what properly written code looked like. Typically these are not online. Consider a wiki page for security coding practices and other reference materials which are easily discovered and readable by people without a security background.

- **Help evaluate security tools**: It is unusual for people outside security to fully understand what security tools do or how they work. So you can help in two ways; first, help developers select tools. Misconceptions are rampant, and not just because vendors over-promise. Additionally it is uncommon for developers to evaluate code scanners, activity monitors, or even patch management system *effectiveness*. In your role as advisor it is your responsibility to help DevOps understand what the tools can provide and what fits within your shared testing framework. Sure, you might not be able to evaluate the quality of the API, but you can tell when a product fails to

deliver meaningful results. Second, you should help position the expenditure, as it's not always clear to the people holding the purse strings how specific tools address security and compliance requirements. You should specify functional and reporting requirements for tools which meet business needs.

- **Help with priorities**: During our research many security pros told us that all vulnerabilities started looking like high priorities, and it was incredibly difficult to differentiate a vulnerability with impact on the organization from one which did not. Exposure analysis is outside the developer skill set. You need to help fill this gap because not every vulnerability poses real risk. And security folks have a long history of sounding like the terrorism threat scale, with vague warnings about "severe risk" and "high threat levels". None of these warnings are valuable without mapping a threat to possible exploitations, what the exploit might mean to the business, or what you can do to address — and reduce — risks. For example you might be able to remediate a critical application vulnerability in code, patch supporting systems, disable the feature if it's not critical, block with IDS or firewalls, or even filter with WAF or RASP technologies. Sometimes code exploitation cannot actually harm the business, so "do nothing" is the right answer! Rather than the knee-jerk "OMG! Fix it now!" reaction we have historically seen, there are typically several options to address a vulnerability, so presenting tradeoffs to a DevOps team allows them to select the best fit.

- **Write tests**: DevOps has placed some operations and release management personnel in the uncomfortable position of having to learn to script, code, and expose their work for public review. It pushes people outside their comfort zones in the short term, but that is a key part of building a cohesive team in the medium term. It is perfectly acceptable for security folks to contribute tests to the team: scans that validate certificates, checks for known SQL injection attacks, open source tools for locating vulnerabilities, and so on. If you're worried about it, help out and integrate unit and regression tests. Integrate and ingratiate! To participate in a DevOps team, where automation plays a key part, you will likely need to know how to write scripts or templates. The good news is that your policies are now embodied into the definition of the environment. Don't be afraid that you don't know source code control or the right format for the scripts — this is an area where developers are usually keen to assist. You need to relearn a bit on the scripting side before your tests can be integrated into build and deployment servers, but you'll do more than preach security — you can contribute!

- **Advocacy**: One crucial area where you can help development is in understanding both corporate and external policy requirements. Just as a CVE entry tells you next to nothing about how to actually fix a security issue, internal security and privacy policy enforcement are often complete mysteries to the development team. Developers cannot Google those answers so offer yourself as an advisor. You should understand compliance, privacy, software licensing, and security policies at least as well as anyone on the development team, so they will probably appreciate the help.

Conclusion

Security's integration with the DevOps movement is difficult. Our last paper, Putting Security Into DevOps, was written to balance the needs of Development, IT and security. This paper is a reaction to market demand, and focuses on the security folks who are tasked with getting security into DevOps, and struggling to integrate — culturally or mechanically — into DevOps. They are the ones who have been asking questions and asking for help. They are the ones having trouble getting the attention of developers, much less developer participation to integrate security tooling and metrics. If you fall into this area, we hope you will find this material helpful. The pain of security testing, and the problem of security controls being outside the domain of developers and IT staff, can be mitigated with DevOps. Security can be largely automated, built slowly over time, and part of the entire release process. And security need no longer be the first casualty of the war for new features and functions — instead it becomes systemized in the delivery process. We have outlined the reasons we expect DevOps to be significant for software development teams in the future, and to advance security testing within application development teams far beyond where it's stuck today.

We are under no illusion that the journey will be easy, as the path from traditional development to DevOps requires tons of hard work while jettisoning long held beliefs about how to build applications. Every organization hits roadblocks and failures along the way, but the good news is the entire approach is geared to assume failures will occur, and help you identify and move past them. The benefits of slow, consistent and demonstrable improvements benefit not just code quality but security; not just developer relations with operations but with security professionals as well.

We hope you find this research helpful. If you have any questions on this topic, or want to discuss your situation specifically, feel free to send us a note at info@securosis.com or ask via the Securosis blog.

About the Analyst

Adrian Lane, Analyst/CTO

Adrian Lane is a Senior Security Strategist with 25 years of industry experience. He brings over a decade of C-level executive expertise to the Securosis team. Mr. Lane specializes in database architecture and data security. With extensive experience as a member of the vendor community (including positions at Ingres and Oracle), in addition to time as an IT customer in the CIO role, Adrian brings a business-oriented perspective to security implementations. Prior to joining Securosis, Adrian was CTO at database security firm IPLocks, Vice President of Engineering at Touchpoint, and CTO of the secure payment and digital rights management firm Transactor/Brodia. Adrian also blogs for Dark Reading and is a regular contributor to Information Security Magazine. Mr. Lane is a Computer Science graduate of the University of California at Berkeley with post-graduate work in operating systems at Stanford University.

About Securosis

Securosis, LLC is an independent research and analysis firm dedicated to thought leadership, objectivity, and transparency. Our analysts have all held executive level positions and are dedicated to providing high-value, pragmatic advisory services. Our services include:

- **Primary research publishing**: We publish the vast majority of our research for free through our blog, and package the research as papers that can be licensed for distribution on an annual basis. All published materials and presentations meet our strict objectivity requirements, and follow our Totally Transparent Research policy.
- **Cloud Security Project Accelerators**: Securosis Project Accelerators (SPA) are packaged consulting offerings to bring our applied research and battle-tested field experiences to your cloud deployments. These in-depth programs combine assessment, tailored workshops, and ongoing support to ensure you can secure your cloud projects better and faster. They are designed to cut months or years off your projects while integrating leading-edge cloud security practices into your existing operations.
- **Cloud Security Training:** We are the team that built the Cloud Security Alliance CCSK training class and our own Advanced Cloud Security and Applied SecDevOps program. Attend one of our public classes or bring us in for a private, customized experience.
- **Advisory services for vendors**: We offer a number of advisory services to help our vendor clients bring the right product/service to market in the right way to hit on critical market requirements. Securosis is known for telling our clients what they NEED to hear, not what they want to hear. Clients typically start with a strategy day engagement, and then can engage with us on a retainer basis for ongoing support. Services available as part of our advisory services include market and product analysis and strategy, technology roadmap guidance, competitive strategies, etc. Though keep in mind, we maintain our strict objectivity and confidentiality requirements on all engagements.
- **Custom Research, Speaking and Advisory**: Need a custom research report on a new technology or security issue? A highly rated speaker for an internal or public security event? An outside expert for a merger or acquisition due diligence? An expert to evaluate your security strategy, identify gaps, and build a roadmap forward? These defined projects bridge the gap when you need more than a strategy day but less than a long-term consulting engagement.

Our clients range from stealth startups to some of the best known technology vendors and end users. Clients include large financial institutions, institutional investors, mid-sized enterprises, and major security vendors. For more information about Securosis, visit our website: <http://securosis.com/>.