Putting Security Into DevOps

Version 1.0
Updated: October 30, 2015

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.

Veracode's cloud-based service and systematic approach deliver a simpler and more scalable solution for reducing global application-layer risk across web, mobile and third-party applications. Recognized as a Gartner Magic Quadrant Leader since 2010, Veracode secures hundreds of the world's largest global enterprises, including 3 of the top 4 banks in the Fortune 100 and 20+ of Forbes' 100 Most Valuable Brands.

Learn more at www.veracode.com, on the Veracode blog and on Twitter.

Copyright

Table of Contents

Introduction

We believe DevOps is one of the most disruptive trends ever to hit software development, and will drive organizational changes over the next decade. But it is equally disruptive for application security, in a good way. DevOps enables developers to weave security testing, validation, and monitoring into both application *development* and *deployment*. To illustrate how this affects application security, this research paper will dive into what DevOps is, and then explain how delivering secure code fits within the larger picture. But to understand why most software development organizations are adopting this trend — often over strenuous objections — you need to understand why it's so attractive, and the problems it is helping organizations solve.

I have been in and around software development my entire professional career. As a new engineer, as an architect, and later as the guy responsible for the whole show. And I have seen as many failed software deliveries — late, low quality, off-target, etc. — as successes. Human dysfunction and miscommunication seem to creep in everywhere, and Murphy's Law is in full effect. Getting engineers to deliver code on time was just one dimension of the problem — the interaction between development and QA was another, and how either could both barely contain their contempt for IT was yet another. Low-quality software and badly managed deployments make productivity go backwards. Worse, repeat failures and lack of reliability create tension and distrust between all the groups in a company, to the point where they become rival factions. Groups of otherwise happy, well-educated, and well-paid people can squabble like a group of dysfunctional family members during a holiday get-together.

When development teams of the past decade tried to go 'Agile' they often ran smack into the other groups within their own firms who remained steadfastly un-Agile. This resulted in more of the same in inter-group friction and further compounded communication and organizational issues. This dysfunction can have a paralytic effect, dropping productivity to nil. Most people are so entrenched in traditional software development approaches that it's hard to see development ever getting better. And when firms who have adopted DevOps talk about deploying code every day instead of every year, or being fully patched within hours, or detection *and* recovery from a bug within minutes, most developers scoff at these notion as pure utopian fantasy. That is, until they see DevOps in action — then their jaws drop.

In this paper we focus on how to put security into the processes of DevOps, as well as the pre-cursor steps of continuous integration and continuous deployment. For readers new to the concept, we will discuss what DevOps is, and then map existing secure development ideas into this new

framework. Our goal is to educate both security professionals who work with DevOps development teams, as well as developers and operational folks who are new to security.

In order to accomplish these goals we will talk about the fundamentals of what DevOps is and how it works. We will not focus on the details of DevOps theory — there are many other great research books and videos on this subject — but rather how to integrate security tools and testing directly into the framework. With DevOps security becomes a natural part of the process, automated in the same fashion as code and platform validation. And in this case, the results are much more than the sum of the parts, as security is simultaneously more effective and less of an impediment.

The Emergence of DevOps

What Is It?

We're not here to dive too deep into the geeky theory behind DevOps — that's outside our focus for this research paper. But as you begin to practice DevOps you will need to delve into its foundational elements to guide your efforts, so we will reference several here. DevOps was born out of lean manufacturing, Kaizen and Deming's principles around quality control techniques. The key concept is continuous elimination of waste — resulting in improved efficiency, quality, and cost savings. There are numerous approaches to waste reduction, but in software development the key concepts are reducing work-in-progress, finding errors quickly to reduce rework costs, scheduling techniques, and process instrumentation so progress can be measured. These ideas have been proven in practice for decades, but mostly applied to manufacture of physical goods. DevOps applies these practices to software delivery, enabled by advances in automation and orchestration.

Theory is great, but how does that help you understand DevOps in practice?

DevOps is an operational framework that promotes software consistency and standardization through automation. Its focus is on using automation to do a lot of the heavy lifting of building, testing, and deployment. Scripts build organizational memory into automated processes to reduce human error and force consistency.

Development, quality assurance, and IT operations teams automate as much of their daily work as they can, investing time up front to make things easier and more consistent over the long haul. The focus is not just on applications, or even an application stack, but on the entire ecosystem. This is commonly referred to as "infrastructure as code" — and is a handy way to think about the configuration, creation, and management of underlying *servers and services* that applications rely on. From code checkin, through code validation, pre-deployment validation, and finally deployment, including run-time monitoring — anything used to get applications into the hands of users is in scope. Automation offers irresistible value in many areas — including scripts and programs to automate builds, functional testing, integration testing, security testing, and even deployment. DevOps aims to make each release a bit faster and a little more predictable than the last. But automation is only half the story, and the less important one for disruption.

Organizational Impact

DevOps represents a cultural change as well, and change in the way an organization behaves has the most profound impact. Development teams focus on code development, quality assurance teams on testing, and operations on keeping things running. Each is a core component of building new services, but **none** are the end goal, just one piece of the whole. In practice these three activities *are not* aligned, and many firms find they are competing priorities, with each group incentives at odds with the other groups. DevOps has Development, QA, and Operations work together to deliver stable applications — efficient teamwork becomes a priority. This subtle change in focus has a profound effect on team dynamics. It removes much of the friction between groups, as they no longer work on 'their' pieces in isolation. It also minimizes many terrible behaviors that cause teams grief: incentives to push code before it's ready, fire drills to fix broken code, deployment issues at release, overburdening key people, *ad hoc* changes to production code and systems, and blaming 'other' groups for systemic failures. Yes, automation is key for tackling repetitive tasks — reducing human error, and enabling people to focus on tougher problems. But DevOps' effect is a bit like opening a relief value — teams work together to identify and address the obstacles that complicate their shared job of producing quality software. By performing simpler tasks more frequently more often, releasing code becomes reflexive. DevOps drives investment in tools — building, buying, and integrating — to achieve better quality, visibility, and ease... which then improve every future release. Success begets success.

Some of you reading this will remark, "That sounds like what Agile development promised", and you will be right. But Agile development focused on the development team; as a result it stumbled at organizations where project management, testing, and IT were not agile. We have seen this defeat many transitions to Agile. DevOps focuses on getting your house in order first, targeting internal roadblocks that introduce errors and slow the process down. Agile and DevOps are complementary, with Agile techniques like scrum meetings and sprints fitting perfectly within a DevOps program. And DevOps ideas on scheduling and use of Kanban boards have morphed into Agile Scrumban tools for task scheduling. They fit together very well.

Problems Addressed

DevOps solves several problems, including those mentioned above. It's time to discuss the specifics in greater detail, and if you had any doubt as the inexorable rise of DevOps in development shops, this section will erase those doubts. Also note that some of the following items overlap. When you are knee-deep in organizational dysfunction, it is often hard to pinpoint the causes. We usually find multiple issues making thing more complicated than they should be, and masking the true nature of the problem. So I want to discuss the problems DevOps solves from multiple viewpoints.

Reduced errors: Automation reduces common errors when performing basic (repetitive) tasks. More than that, automation is valued for preventing *ad hoc* changes to systems, which are often used instead of complete documented fixes. In the worst case the problem and solution are both undocumented and the underlying issue is never actually fixed, and is not much more than the fleeting memory of the person who fixed the issue in a panic during the last release. By including configuration and code updates within automation, settings and procedures are applied consistently

— every time. Incorrect settings are addressed in automation scripts, fixes are pushed into production, and change logs are available — *ad hoc* alterations are avoided.

Speed and efficiency: Here at Securosis we talk a lot about "reacting faster and better" and "doing more with less". DevOps, like Agile, is geared towards doing less, better, and faster. Releases occur more regularly, with less code change between them. Less work means better focus, and more clarity of purpose with each release. Again, automation helps people get their jobs done with less hands-on work. But it also helps speed things up: *The Phoenix Project* — a landmark novel about DevOps — equates this to moving from cannon fire to anti-aircraft fire. Software builds can occur at programmatic speeds. Orchestration scripts that can spin up build and test environments on demand, there is no waiting around for IT to provision systems — it's all built into the automated process. If an automated build fails, scripts can pull the new code and alert the development team. If automated functional or regression tests fail, the information is in QA's or developers' hands before they finish lunch. You can fail faster, with turnaround to identify and address issues quicker as well.

Bottlenecks: There are several bottlenecks in software development: developers waiting for specifications, select individuals who are overtasked, provisioning IT systems, testing, and even processes (particularly synchronous ones, as in waterfall development) can all cause delays. The way DevOps tasks are scheduled, the reduction in work being performed at any one time, and the way expert knowledge is embedded into automation, all act to reduce these issues. Once DevOps is established it tends to alleviate major bottlenecks common to most development teams, especially the over-burdening of key personnel.

Cooperation and communication: If you have ever managed software releases, you have witnessed the ping-pong match between development and QA. Code and insults fly back and forth between groups — when they're not complaining about how long it's taking IT to get things patched and new servers available for testing and deployment. The impact of having Operations, Development, and QA work shoulder to shoulder is hard to articulate, but focusing the teams on smaller sets of problems to address *in conjunction* reduces friction around priorities and communication. You may consider this a 'fuzzy' benefit... until you see it firsthand, and realize how many problems are alleviated by clear communication and shared purpose.

Technical Debt: Most firms consider the key function of development to be producing new features for customers. Things that developers want — or need — to produce more stable code are *not* features. Every software development project I have ever participated in ended with a long list of things we needed to do to improve the work environment (the "To Do" list). This was separate and distinct from new features: new tools, integration, automation, updating core libraries, addressing code vulnerabilities, and even bug fixes. Project managers ignore them as not *their* priority, and developers fix issues at their own peril. This list is the essence of technical debt, and it piles up fast. DevOps attempts to reverse priorities and targets technical debt — and anything else that slows down work or reduces quality — *before* adding new capabilities. The 'fix-it-first' approach produces higher-quality and more reliable software.

Metrics and Measurement: Are you better or worse than you were last week? How can you tell? The answer is metrics. DevOps is not just automation, but also continuous and iterative improvement. Collection of metrics is critical for knowing where to focus your attention. Captured data from platforms and applications forms the bedrock for measuring everything — from quantifiable aspects like latency and resource utilization, to more abstract concepts like code quality and testing coverage. Metrics are key for knowing what's working and what needs improvement.

Security: Security testing — just like functional testing, regression testing, load testing, and just about any other form of validation — can be embedded into the process. Security becomes not just the domain of security experts with specialized knowledge, but integrated into the development and delivery process. Security controls can be used to flag new features or gate releases — within the same set of controls you use to ensure custom code, application stacks, or server configurations, meet specifications. Security goes from being "Dr. No" to just another set of tests which help validate code quality.

Security Integration Points

Remember, DevOps is about joining Development and Operations to provide business value. The mechanics of this are incredibly important; it drives how the teams work together. And it's how they become a single — focused — effort. In the same way, security should be part of the development and operational framework. There is no 'SecDevOps', it's just DevOps. Let's discuss why that is.

The Basics

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. 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.

Continuous Integration

The essence of Continuous Integration (CI) is developers regularly checking 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 these tests commence automatically whenever a new bundle is available. 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. This

orchestration does not happen overnight — it is an evolutionary process that typically take months to establish the basics and years to mature.

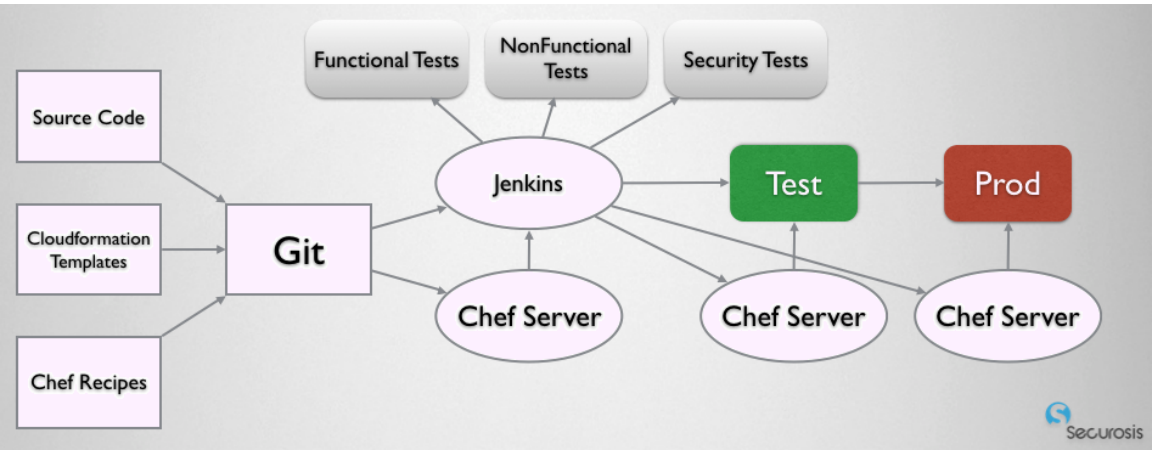
Continuous Deployment

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. Rich Mogull created the following graphic to show the flow of code, from check-in to deployment, and many of the tools that support automation.

Upon a successful completion of a CI cycle, the results feed the Continuous Deployment (CD) process. CD takes another giant step forward for automation and resiliency. By building infrastructure automatically, and in the same way each time new code is committed, the test environment and the deployment environment become mirror images. And deployment validation can occur on essentially the same image. But where things get really wild is when firms opt to *release code automatically to end users* once it's completed the battery of pre-deployment tests! Not all firms do this, but some notable companies like Netflix, Google and Etsy have automated releases once their tests have completed.

CD addresses dozens of issues that plague code deployment — particularly error-prone manual changes, and discrepancies in revisions of supporting libraries between production and development. But perhaps most important is use of code and infrastructure to control deployment and rollback in case of errors. We will go into more detail in the following sections.

DevOps In Action



As developers check in code, scripts automatically compile and tools like Jenkins build the application. Tools like Chef, Puppet and Ansible not only automate the building of infrastructure, but they also are used to enforce security policies. Scripts may pull certified 'golden masters', or they can automatically assemble secure application stacks by updating, patching, configuring and self-validating. They not only can build the application stack, but assemble test tools, populate test

databases, and then run security tests — commonly many tests in parallel. At each step along the way, errors at any level are immediately reported, and sent to DevOps team, with faulty changes automatically backed out of code pending investigation. Should all of the test complete, notice that there is little difference between the automated building and configuration of test servers from production servers; this is how some organizations continuously deploy code changes, possibly dozens of times per day. It's all made possible through the automation of traditional development, quality assurance and IT processes.

This is far from a complete description, but hopefully you get the basic idea of how it works. With the basics of DevOps in mind, let's now map security in.

Mapping to an SDLC

Secure Development Lifecycle's (SDLC), also called Secure Software Development Lifecycle's, describe how security fits into the different phases within software development. Looking at the different phases in an SDLC, most developers see a pre-Agile waterfall development process, which makes SDLC look like a poor fit with DevOps. But there are good reasons to *conceptually* unite SDLC with DevOps: SDLC's architecture, design, development, testing, and deployment phases all map well to development roles *regardless of development process,* so they provide a good jumping-off point for people to adapt current models and processes into a DevOps framework.

Define

Operational Standards: The early phases of software development typically focus on the big picture of application architecture, and how large functional pieces will work. With DevOps you are also weaving in operational standards for the underlying environment. Just as with the code you deploy, you want to make small iterative improvements to your operational environment every day. This includes infrastructure updates such as build automation and CI tools, as well as application stack policies — including security, how patches are incorporated, version synchronization across the build chain, leveraging tools and metrics, configuration management, and testing. These standards will shape the stories sent to Operations for scripting during the development phase, discussed below.

Security Requirements: Just as with minimum set of functional tests which must be run prior to code acceptance, you'll have a set of security tests will you run prior to deployment. These may be an agreed upon battery of unit tests for specific threats your team writes. Or it may include addressing all critical issues found in SAST testing. And what you test, and the tools you employ, is entirely up to you. For example, you may require all OWASP Top Ten vulnerabilities be mitigated in code or supporting products. Regardless of what you choose, the baseline requirements should account for new functionality as well as old. A growing body of tests will require more resources for validation and — possibly — slow your test and deployment cycle over time, so you have some decisions to make regarding what 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? What will slow down? What is working and how can you prove it? Metrics are key to

answering all these questions. You will need to think about what data you want to collect, and build it into your CI and CD environment to measure how your scripts and tests perform. You'll continually evolve your collection and use of metrics, but plan for basic collection and dissemination of data from the get-go.

Design

Secure Design/Architecture: DevOps enables significant advancements in security design and architecture. Most notably, since your goal is to automate patching and configurations for deployment, you can entirely disable administrative connections to production servers. Errors and misconfigurations are fixed in build and automation scripts, instead of through manual logins. Configuration, automated injection of certificates, automated patching, and even pre-deployment validation are all possible. You can also completely disable administrative network ports and access points, eliminating a common attack vector. APIs from PaaS and IaaS cloud services offer even more automation choices, as we will discuss later in this paper. DevOps offers a huge improvement to basic system security, but you must specifically design or re-design your deployments to leverage the advantages which automated CI and CD can provide.

Secure the Deployment Pipeline: With both development and production environments more locked down, development and test servers become more attractive targets. Traditionally these environments run with little or no security. But there is greater need for secure source code management, build servers, and deployment pipelines, given their much stronger (automated) links to production. You'll need stricter access controls for these systems, particularly build servers and code management. And given scripts running continuously in the background with minimal human oversight, you'll need additional monitoring to catch errors and misuse. When deployed on virtualized or especially cloud environments, where the management plane allows for control of your entire virtual data center, great care needs to be taken with who has access, and tracking of changes made to templates and build scripts.

Threat Model: We maintain that threat modeling is 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 them plan out unit tests to address such attacks. Threat modeling is often performed during design phases, but can be done as smaller units of code are developed as well, and enforced with unit tests or systemic testing.

Develop

Infrastructure and Automation First: You need tools before you can build a house, and you need a road before you drive a car somewhere. With DevOps, and particularly with DevOps security, integrating tools and building tests are performed *before* you begin developing the next set of features. We stress this because it makes planning more important, and because it helps Development plan for the tools and tests they need to deploy before they can deliver new code. The bad news is that there is up-front cost and work to be done; the good news is that each and every build now leverages the infrastructure and tools you built.

Automated and Validated: Remember that not only Development is 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 that build out the infrastructure needed for development, testing, and production servers. The good news is that what works in testing should not change production. And automation helps eliminate the problem traditional IT has faced for years: *ad hoc* undocumented work that runs months — or even years — behind on patching. Again, there is a great deal of work to get this fully automated — on servers, network configuration, applications, and so on. 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.

Security Tasks: 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.

Security in the Scrum: As mentioned in the last section, DevOps is process neutral. You can use spiral, or Agile, or surgical-team approaches, as you wish. But Agile Scrums and Kanban techniques are well-suited for DevOps. Their focus on smaller, focused, quickly demonstrable tasks aligns nicely with DevOps. For security, these tasks are no less important than any other structural improvement. We recommend training at least one person on each team on security basics, and determining which team members are interested in security topics, to build in-house expertise. This way security tasks can easily be distributed to team members with interest and skill in tackling them.

Test

Strive for Failure: DevOps turns many long-held principles — of both IT and software development — upside down. Durability used to mean 'uptime', but now it's speed of replacement. Detailed specifications to coordinate dev teams have been replaced by Post-It notes. Quality Assurance focused on getting code to pass functional requirements, but now they look for ways to break an application before someone else can. This new approach helps to raise the security bar. 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 test functions during the automated delivery process, but really test the ruggedness of the code, and substantially raise the minimum quality of an acceptable release. We harden an application by intentionally pummeling it with all sorts of functional, stress, and security tests *before* code goes live, reducing the time requirement for hands-on security experts testing code. If you can figure out some way to break your application, odds are attackers can too, so build the test — and the remedy — before the code goes live.

Parallelize Security Testing: A problem common to all Agile development approaches is what to do about tests that take longer than the development cycle. For example we know that fuzz testing critical pieces of code takes longer than an average Agile sprint. 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 or dynamic code scanning. To address this issue DevOps teams run

multiple security tests in parallel to shorten the test window. Validation against known critical issues is written as unit tests for quick spot checks, with failures kicking code back to the Development team. Code scanners are typically run in parallel to unit or other functional tests. These results are also sent back to Development, and identify the changes that created the vulnerability. Organizing tests for efficiency vs. speed — and completeness vs. time to completion — was an ongoing balancing act for every dev team we spoke with. Focusing scans on specific areas of code helps find issues faster. Several firms also discussed a move to have pre-populated and fully configured tests servers — just as they do with production servers — waiting for the next test sweep to avoid latency. Rewriting and reconfiguring test environments for efficiency and quick deployments will help with CI efforts.

Deployment

Manual vs. Automated deployment: It is easy enough to push new code into production. Vetting that code, or rolling back in case of errors, is much harder. Most teams we spoke with are not yet fully comfortable with fully automated deployment. In fact many still only release new code to customers every few weeks, often in conjunction with the end of a sprint. These companies execute most deployment actions through scripts, but they launch the scripts manually when Operations and Development resources are available to fully monitor the push. A handful of organizations are comfortable with fully-automated pushes to production, and release code several times a day. There is no right answer here, but in either case automation performs the bulk of the work, freeing personnel up 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 typically saw three tricks used to augment deployment. The first, and most powerful, of these techniques is called Blue-Green or Red-Black deployment. Old and new code run side by side, each on their own set of servers. A rollout is a simple redirection at the load balancer level, and if errors are discovered, the load balancers are pointed back to the old 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 (meaning 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. In the event errors are discovered in a new section of code, the feature can be toggled off until the code is fixed. The degrees of automation and human intervention vary greatly, but overall these deployments are far more automated that traditional web services environments.

Production Security Tests: Applications often continue to function even if security controls fail. For example, a new deployment scripts may miss the update to web application firewall policies. Validation — or at least sanity checks on critical security pieces — is essential in the production environment. The good news is it is fairly easy to have the 'canaries' be dynamic code scanners, penetration testers, or other security testers to vet security controls are in place and functioning.

Security Tools for DevOps

In this section we show you how to weave security into the fabric of your DevOps framework. DevOps encourages testing in all phases of development and deployment. Better still, it easily accommodates security testing side by side with functional and regression tests. From each developer's desktop prior to check-in, to module testing, and eventually against a full application stack, both pre- and post- deployment — it is all available.

Where To Test

Unit Testing: 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 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 — but 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.

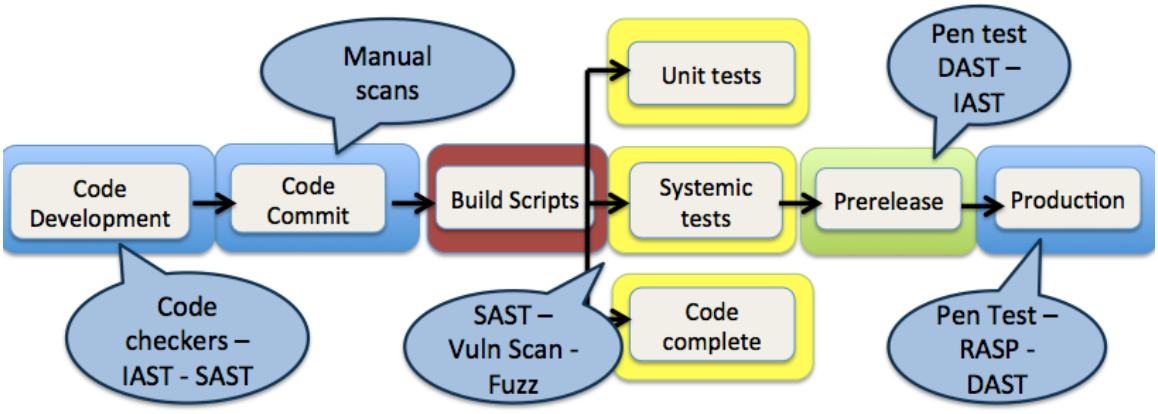
Production Runtime Monitoring: To understand what's working and what is not, DevOps requires much better monitoring and metrics than what's commonly provided with legacy platforms. To validate that the new version is actually working, monitoring and instrumentation needs to be built in. You need metrics at the speed of DevOps! Most organizations will focus on performance metrics to

ensure new code offers reasonable performance, and even rely upon big data clusters like Hadoop or Splunk to aid application analysis. And these same ideas should be extended to security as well, monitoring and logging changes to the environment that are used to change applications and the supporting infrastructure. As mentioned earlier, many organizations now take advantage of Blue-Green deployments to run tests of all types against new production code. 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. But how do you know the older versions of code not longer support user requests, or how do you provide change management reports for compliance? Production monitoring is not only key for understanding application performance and operational issues, but compliance and security as well.

Managed Releases: Balancing thoroughness against timelines is a battle for most organizations. The goal is to test and deploy quickly, with many organizations who embrace CD releasing new code a minimum of 10 times a day. Quality and depth of testing become more important. If you have massaged your CD pipeline to deliver every hour, how can you take advantage of static or dynamic scans that take several hours to several days to produce results? This is why some organizations do *not* use automated releases — instead they wrap releases into a 'sprint', running a complete testing cycle against the results of the last development sprint. Others take periodic snapshots of their code and run white box tests in parallel, gating their releases on those results, addressing any issues revealed with new task cards. Another way to look at this problem is that, just like all your Dev and Ops processes, testing and deployment procedures *themselves* will go through iterative and continual improvement — your definition of 'done' for pre-release security testing need continual improvement, just like everything else. You may add more unit and regression tests over time, and shift more load to developers, before they check code in.

Building A Tool Chain

Here we offer a list of common security testing techniques, the value they provide, and where they fit into a DevOps process. Many of you already appreciate the value of tools, but not necessarily how they fit in a DevOps framework, so we will contrast traditional vs. DevOps deployments. Odds are you will use many, if not all, of these approaches — broad testing helps to find code weaknesses, and determine whether issues genuinely threaten application security.



The above graphic reflects our conversations, with development and security practitioners, on where they are successfully deploying security testing tools in a DevOps framework. The callouts map the types of tests being conducted at specific phases of CI & CD. Keep in mind that it's early days for DevOps and the orchestration of security tools — basically what works where — is far from settled. More importantly, many security tools were built before these concepts of rapid and automated deployment existed; older products are too slow, some could not focus their tests on new code, and still others did not offer API support. Which is another way of saying not all tools are created equal, so you'll need to evaluate for both performance and API integration capabilities as well as code coverage capabilities.

Let's dive into the different types of testing tools available:

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. 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. 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'.

Fuzzing: At its simplest fuzz testing is essentially throwing lots of random garbage at applications, seeing whether any particular (type of) garbage causes errors. 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 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 don't have copies of the code, but much more problematic for applications delivered to users (including mobile apps, desktop

applications, and automobile systems). This trend worries us — like penetration testing, periodic fuzz testing should be part of your security testing efforts. Fuzzing may be part of unit tests, or part of QA's parallel testing.

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. 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 toolbelt should include manual code inspection — at least periodic spot checks of new code.

Vulnerability Analysis: 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. Some even use credentials to query the application for detailed information. And there are tools that actively probe an application looking for poorly implemented code, such as how user credentials are handled. Make sure you broaden you scans to include your application, your application stack, and the platforms that support it.

Version Controls: One of the nice side benefits of build scripts running both QA and production infrastructure is that Dev, Ops, and QA are all in synch on the versions of code they use. But someone on your team still needs to monitor and control versions and updates for all parts of the application stack. For example, are all your gems up to date? As with vulnerability scanning, you should monitor your open source and commercial software for new vulnerabilities, and create task cards for patches to the build process. But many vulnerability analysis products don't cover all the bits and pieces that comprise an application. This can be fully automated in-house, with scripts adjusted to pull the latest version, or you can integrate third-party tools for monitoring and alerting. Either way version control should be part of your overall security monitoring program, with or without vulnerability analysis.

Runtime Protection: This is a new segment of the application security market. The technical approaches are not new, but over the past couple years we have seen greater adoption of security tools embedded into applications for runtime threat protection. These tools are called by different names, including Runtime Application Self Protection (RASP) and Interactive Application Self-Testing (IAST) depending on the specific variation; essentially they provide execution path scanning, monitoring and embedded application white listing. So do the deployment models (including embedded runtime libraries, in-memory execution monitoring, and virtualized execution paths), but they all attempt to protect applications by detecting attacks in runtime behavior. These platforms can

all be embedded into the build or runtime environment; they can all monitor or block; and they all offer adjustable enforcement, based upon the specifics of the application. While these technologies are relatively new, they fill a gap in existing application security validation and protection.

Priorities and Risk

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 risk? If it is a risk rather than a false positive, what is its priority, relative to everything else? How is this information distributed? With DevOps you need to close the loop on issues within infrastructure, 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 mentioned in an earlier section, DevOps is not all about tools and technology, but much of its success is how people work within this model. In this section we outline how security folks should consider their role in a DevOps environment. A while back we posted a research paper on Putting Security Into Agile Development. We received feedback that the most helpful part of that report was guidance for security people on how best to work with Development. People appreciated advice on how best to position security to help development teams be more Agile, and as DevOps can encompass and extend Agile concepts across an entire organization, here we provide similar examples for the role of security.

Keep in mind, there really is no such thing as SecDevOps. The beauty of DevOps is that security becomes part of the operational process of integrating and delivering code. We don't call security out as a separate thing because it is not actually separated — instead it is (at least should be be) woven into the DevOps framework. Security professionals should keep this in mind when considering how they fit within the new development framework. You will need to play one or more roles in the DevOps model of software delivery, so look at how you can improve delivery of secure code without waste and without introducing bottlenecks. The good news is that security fits nicely within this framework, but you need to tailor your approach to work within the automation and orchestration model to be successful.

The CISO's Responsibilities

Learn the DevOps process: To work in a DevOps environment you need to understand what it is and how it works. You need to understand what build servers do, how test environments are built, what it means to dispense with manual intervention, the automated nature of workflows, and what gates each step in the process. Find someone on the team; have them walk you through the process and introduce the tools. Once you understand the process, the security integration points should become clear. Once you understand the mechanics of the development team, you'll have a better idea of how to work with them, in context of 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 that offer quick feedback. You need to adjust requirements and recommendations so they fit into the process, often simplified into small steps, with enough information for the 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.

How CISO's Support DevOps

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 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: Security teams are typically small and often lack budget. What's more, security people are not present at most development meetings, so they lack visibility in day-to-day DevOps activities. To help extend the reach of the security team, see if you can get someone on each development team to act as a security advocate. This helps not only extend the security team's reach, but also expand security awareness throughout the development process.

Help DevOps team understand threats: Most developers don't fully grasp how attackers approach attacking a system, or what it means when a 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 that 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. Real-world use cases go a long way to help developers and IT understand why it is essential to protect application functions from certain threats.

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.

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 capabilities. Additionally it is uncommon for developers to evaluate code scanners, activity monitors, or even patch management systems. In your role as advisor it is your responsibility to help DevOps understand what the tools can provide and what fits within your 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 the tool that meet the business needs.

Help with priorities: 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, 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. 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 for their systems.

Write tests: DevOps has placed some operations and release management personnel in the uncomfortable position of having to learn to script, code, and place their work open 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. Know this: everyone writes bad code from time to time, which is why we need to focus on validation prior to release. 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! You may need to learn a bit on the scripting side before your tests can be integrated into the 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

Software developers traditionally do not embrace security. It's not because they do not care about security — but historically they have been incentivized to to focus on delivery of new features and functions. Security tools don't easily integrate with classic development tools and processes, often flooding development task queues with unintelligible findings, and lack development-centric filters to help developers prioritize. Worse, security platforms and the security professionals who recommended them have been difficult to work with — often failing to offer API-layer integration support or architectures that fit within new development frameworks.

DevOps may be new to software development, but it's not new to manufacturing; these concepts have been proven to be effective for several decades and remain in use around the globe. For software development teams to embrace this approach offers many potential advancements for the timeliness and quality of the entire application stack, and that includes security. That said, security integrates with DevOps only to the extent that development teams build it in. Automated security testing, just like automated application building and deployment, must be factored in along with the rest of the infrastructure.

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. 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 or 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 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:

- **The Securosis Nexus**: The Securosis Nexus is an online environment to help you get your job done better and faster. It provides pragmatic research on security topics that tells you exactly what you need to know, backed with industry-leading expert advice to answer your questions. The Nexus was designed to be fast and easy to use, and to get you the information you need as quickly as possible. Access it at <https://nexus.securosis.com/>.

- **Primary research publishing**: We currently release the vast majority of our research for free through our blog, and archive it in our Research Library. Most of these research documents can be sponsored for distribution on an annual basis. All published materials and presentations meet our strict objectivity requirements and conform to our Totally Transparent Research policy.

- **Research products and strategic advisory services for end users**: Securosis will be introducing a line of research products and inquiry-based subscription services designed to assist end user organizations in accelerating project and program success. Additional advisory projects are also available, including product selection assistance, technology and architecture strategy, education, security management evaluations, and risk assessment.

- **Retainer services for vendors**: Although we will accept briefings from anyone, some vendors opt for a tighter, ongoing relationship. We offer a number of flexible retainer packages. Services available as part of a retainer package include market and product analysis and strategy, technology guidance, product evaluation, and merger and acquisition assessment. Even with paid clients, we maintain our strict objectivity and confidentiality requirements. More information on our retainer services (PDF) is available.

- **External speaking and editorial**: Securosis analysts frequently speak at industry events, give online presentations, and write and/or speak for a variety of publications and media.

- **Other expert services**: Securosis analysts are available for other services as well, including Strategic Advisory Days, Strategy Consulting engagements, and Investor Services. These tend to be customized to meet a client's particular requirements.

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.

Additionally, Securosis partners with security testing labs to provide unique product evaluations that combine in-depth technical analysis with high-level product, architecture, and market analysis. For more information about Securosis, visit our website: <http://securosis.com/>.