Secure Agile Development

Version 1.0
Updated: November 3, 2014

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 is licensed by Veracode:

Veracode delivers the most widely used cloud-based service for securing web, mobile, legacy and third-party enterprise applications. By identifying critical application-layer threats before cyberattackers can find and exploit them, Veracode helps enterprises deliver innovation to market faster – without sacrificing security. Veracode's powerful cloud-based platform, deep security expertise and programmatic, policy-based approach provide enterprises with a simpler and more scalable way to reduce application-layer risk across their global software infrastructures.

Veracode secures hundreds of the world's largest global enterprises, including 3 of the top 4 banks in the Fortune 100 and more than 25 of the world's top 100 brands. Learn more at www.veracode.com, on the Veracode blog and on Twitter.

Copyright

This report is licensed under Creative Commons Attribution-Noncommercial-No Derivative Works 3.0.

http://creativecommons.org/licenses/by-nc-nd/3.0/us/

Secure Agile Development
Table of Contents

Introduction 4

Agile for the CISO 7

Bringing Security to the Agile Process 10

Building a Security Tool Chain 14

Integrating Security with Deployment Pipelines and DevOps 19

Summary 23

About the Analyst 24

About Securosis 25

Introduction

Over the past 15 years, the way we develop software has changed completely. Development processes evolved from Waterfall, to rapid development, to extreme programing, to Agile, to Agile with Scrum, to our current darling: DevOps. Each evolutionary step was taken to build better software by improving the software building *process*. And each step embraced changes in tools, languages, and systems to encourage increasingly agile processes, while discouraging slower and more cumbersome processes.

The fast flux of development evolution gradually deprecated *everything* that impeded agility… including security. Agile had an uneasy relationship with security because its facets which promoted better software development (in general) broke existing techniques for building security into code.

There are several areas of conflict which, on the surface, make it difficult to embed security into Agile development processes. That isn't because developers don't care about security — most developers we speak with are interested in security and want to address security issues. But their job is delivering functioning code on schedule, and anything else is secondary. Agile frameworks are the new foundation for code development, with an internal focus on ruthlessly rooting out tools and techniques that don't fit the model. So secure development practices, just like every other facet of development, must fit *within* the Agile framework — not the other way around.

This paper is for security professionals who want to understand Agile development and the issues developers face, so both teams can work together better. We will discuss rapid development process evolution, feature prioritization, and how cultural differences can create friction between security and development. Speed and agility are essential to *both* sides — tools and processes that allow security issues to be detected and recovered from earlier benefit everybody.

The Evolution of Agile Development

In the simplest terms, Agile software development is a set of techniques for building intended software with fewer errors and better predictability. Each technique or approach is intended to address well-known weaknesses in traditional 'Waterfall' development: complexity, poor communication, and infrequent code validation.

So Agile approaches embrace task simplicity, fast turnaround for smaller pieces of working code, and human interaction over email/document/process oriented interaction. Simpler tasks make it harder for developers to misunderstand what the code is *meant* to do. Faster iteration produces working code more often, with three distinct benefits: early confirmation that you are building the correct solution, quicker detection of breakage, and more accurate project completion estimates.

Face-to-face human interaction helps clarify what everyone on the team is building and greatly reduces communication errors — far too common with ambiguous documentation and code specifications. Smaller, simpler, and faster.

The most popular flavor of Agile today, by a large margin, is Agile with Scrum. Each characteristic of this approach has been selected to support agility. For example short development 'sprints' — typically 1-4 weeks — are used rather than the 18-month cycles common to traditional Waterfall development. Agile with Scrum relies on daily 'scrum' meetings, where all developers meet face-to-face to discuss what they will be working on that day. Developers receive tasks on 'task' cards — think of a post-it note — which effectively limit task complexity. Each sprint focuses on iterative improvements rather than complete feature delivery. This ensures that only functional code modules are checked in — unlike Waterfall where every feature tends to get crammed in on deadline. Ken Schwaber's book Agile Project Management with Scrum is the best reference we have found, so pick up a copy if you want detailed information.

But why do security professionals care? Because development has shifted focus to smaller, simpler, and faster — excluding slow, indeterminate, and complex security tasks. To get secure development embraced by developers, you need to overcome several issues.

1. **Tests that do not fit the Agile process:** The classic scenario is development teams moving to a 2-week Agile 'sprint', while security counts on 2 months of automated fuzz testing before every release.

2. **Security data that does not integrate with development systems:** The cliché is external penetration testers identifying thousands of instances of hundreds of vulnerabilities, then dumping unexplained findings onto development teams who have no idea what to do with the results.

3. **Security tools that do not fit development realities:** Classic testing tools may require 100% of code, including all supporting libraries, to be fully built before tests can be conducted. This prerequisite breaks small iterative code changes delivered weekly or daily.

4. **Insufficient knowledge:** Even today, many developers do not understand XSS or CSRF, and when they are familiar with these issues it remains unclear how these issues should be addressed across the code base. Understanding threats and remediation were uncommon skills.

5. **Getting security into the work queue:** The move from Waterfall to Agile meant the removal of specific security testing phases, or 'gates', in the Waterfall process. For security features or fixes to be integrated they must now be documented as *tasks* and prioritized over other features — otherwise security gets 'starved' off the development queue.

6. **Policies:** A big book of security policies, vulnerabilities, and requirements is extremely un-Agile. One of Waterfall development's most serious failings is its dependence on large complex specifications that confuse developers. Agile development is an effort to protect developers from such confusing and unwieldy presentation of essential information.

During this amazing period of advancement in software development, we have seen an equally amazing evolution among hackers and attacks against applications of all sorts. Security has largely remained static, but there are plenty of ways to integrate security into software development, provided you are willing to make the necessary adjustments. The following section outlines how to integrate security people — you — and security testing into an Agile process.

Waterfall

Waterfall is a development process that starts with extensive documentation of feature and function requirements, before moving into each subsequent phase required to build, test and ship a final product. Waterfall development often takes 12-24 months to produce a final product, and was derived from construction and manufacturing processes designed to produce physical products, with minimal deviation change from specifications to reduce faults and avoid project overruns.

But Waterfall development is no longer well-suited to most software projects, due to long feedback cycles and rigidity. For example bug fixes — especially time-sensitive security patches — don't fit well into Waterfall's extended timelines.

Agile for the CISO

Why can't we all get along?

There is no point trying to ignore the elephant in the room. Everyone knows there has historically been friction between security professionals and development teams. This isn't because of inherent animosity, but conflicting priorities. Development needs to ship functioning code on time and within budget. Security needs to manage risks to the organization, including risks introduced by new technologies such as code. One needs to go as fast as possible; the other needs to keep from smashing through the guardrails and flying off the road.

Unfortunately sometimes this isn't expressed in the most… professional… of ways. Development teams accuse the CISO of having no appreciation for the skill with which they balance competing priorities, and view security practitioners as noisy know-it-alls who periodically show up shouting "All your stuff's broken!" CISOs don't understand how development teams work, accuse developers of having no clue how attackers will break their code, and suspect the only English phrase they learned in college was "We don't have time to fix that!" This mutual lack of understanding makes even basic cooperation difficult.

We are here to help you understand development issues, and how to communicate what needs to happen to developers without putting them on the defensive. The following are several aspects of Agile development that create friction between these groups. Avoiding these pitfalls will make your job much easier, but you need to put in some effort to understand how development teams work, and how best to work *with* them.

Agile 101 for CISOs

Before we make specific recommendations, *you* need to embrace Agile itself. Agile is fascinating because it isn't a single development process, but a collection of methods under a common banner. That banner is *incremental improvement*. It promotes faster and less complex development cycles, with a focus on personal interaction between developers, working software over documentation, more collaboration with customers, and responsiveness to change. But the key is making small improvements every sprint to improve speed and effectiveness. This is all spelled out in the Agile Manifesto.

You will need to work with the people who control the Agile process. Agile developers have specific roles. A **Product Owner** is responsible for overall product requirements and priorities. The

Development Team builds the code. The **Scrum Master** facilitates frequent structured meetings (**Scrums**), typically daily for 15 minutes, to formally communicate project status.

Projects break down into **Sprints** which are one-to-four week coding cycles with specific tasks to be performed (not necessarily features to be developed). The **Sprint Planning Meeting** defines goals, which are placed into a **Backlog** of tasks for the Program Manager to prioritizes, typically in a ticketing and task management tool. Developers then focus on their assigned small tasks (2-6 hours). Big-picture features are called **Stories**, which are also broken down into tasks.

Each day a developer comes in, attends the scrum, loads up his or her task list, codes, and commits the code when the assignment is complete… so it can run through testing. Code is typically committed near the end of the sprint — but in some versions of Agile, including DevOps, developers commit several times per day.

The key is that sprints are short — usually a matter of weeks. Requirements are set up front, and then everyone is off to the races to knock out particular tasks. The goal is to completely deliver a feature or function at the end of the sprint, getting it into the hands of customers as quickly as possible. This is important for quick customer feedback, and avoids spending 2 years on a feature, only to find out it is the wrong thing.

Recommendations for the CISO

There are several things to do, and to avoid, as you work with development. They mostly come down to integrating security as seamlessly into their processes as possible, without sacrificing security objectives.

- **Learn:** You learned to work with HR on privacy issues, Legal on breach disclosures, and IT on compliance reporting and controls. You know how to work with executive management to communicate risk and ask for budget. Now you need to take time to learn how development works — this research is just a starting point. Focus on their basic development process, and the tools they use to manage it. Once you understand the process the security integration points become clear. Once you understand the mechanics of the development team, the best way to introduce and manage security requirements also becomes evident.

- **Communicate:** We strongly recommend joining the team directly to hash out issues, but only when you need to. Agile promotes individual interaction as a principal means of communication, but that does not mean you should attend every scrum. In today's development suite tools like JIRA, Slack, and even Skype facilitate communication across development teams. Add yourself to the appropriate groups within these communication services to stay abreast of issues, and join meetings as necessary.

- **Grow your own support:** Every development team has specialists. One member may know UI and another mobile platforms; one may be responsible for tools, another build management, and another architecture. During our most recent interviews a handful of companies explained that they encourage each team to have a security specialist to advocate.

- **Provide security training:** Training is expensive, so it is essential for development to leverage lower-cost knowledge sharing options. Most organizations do in-house training such as “lunch and learn”. Whenever a development team member learns something new or brings ideas back from a conference, they present it over pizza (provided by the company to encourage attendance). This is a great way to get in front of developers, educate them on security topics and — most important — answer questions on how to address issues. You might even consider sharing some budget to get key people the education or tools they need.

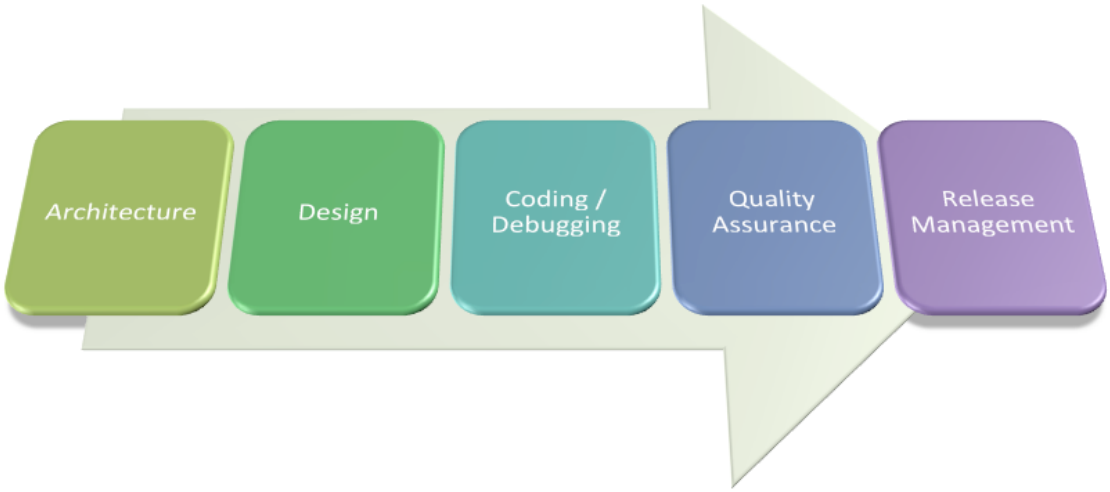
- **Tell your story:** A *story* is Agile’s fundamental unit for communicating project requirements. You may need to get your hands dirty, write security ‘stories’, and help develop task cards to describe security work to be done. Product managers don’t know security all that well, so if you really want the development team to understand what you need built, you will need to help write stories.

- **Advise:** 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 set yourself up as an advisor on these issues. 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.

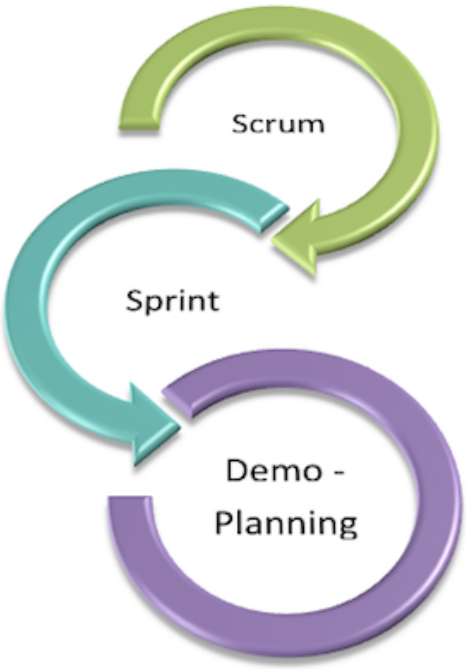
Bringing Security to the Agile Process

One of the toughest parts of bringing security into an Agile program is process modification. The common Waterfall development process has cleanly delineated phases, each of which provides an opportunity for security integration, and each security activity must be completed before moving on to the next phase. Agile includes whatever work gets done in a sprint — it does *not* bend to security, so you need to bend security to fit Agile.

The easiest way to grasp the process changes is to contrast a Waterfall process against Agile methods. This diagram shows a Waterfall development process, listing the sequence of steps, each performed by a different team.



Each Waterfall step includes one or more security tasks. For example the Design phase includes threat modeling to see which relationships are subject to which types of threat. During development we perform static analysis or regression testing for specific vulnerabilities, Quality Assurance includes fuzzing and exception testing, and Release Management sets firewall policy and patches the deployment environment. Each of these security tasks must be completed before the next phase can begin.



Here is a simple Agile with Scrum process diagram. The scrum interval is one day, and a sprint is typically 2-4 weeks. At the end of the sprint the code *milestone* is demonstrated, and the next highest priority tasks on the Backlog are assigned.

Following are several recommendations for how to integrate specific security tasks into Agile. Agile cycles (including Scrum, Sprint, and Planning) are short, and there is no direct mapping from Agile cycles to Waterfall stages. Below we recommend integration points for specific security controls that Agile has largely left behind, in an effort to reinstate tests that were optimized out of the Agile process. We encourage you to think about what works for you, keeping in mind that companies often have multiple development teams, and each may implement Agile differently.

Architecture and Design

The architecture and design phases specify intended behavior: which pieces of code are responsible for which tasks (functions), and the handoffs between different modules (communication and trust). This is where you set security policies, define intended and unacceptable behaviors, and model threats. Agile has no direct equivalent to Waterfall's architecture and design phases so you need to determine the best fit based on your team's skills. In some cases threat modeling is defined by a user story, and becomes a task for architects or security team members who support development operations. In rare cases firms have product managers or architects model threats — provided they've been given training — as the stories are written, baking the results into design changes. Both methods have pros and cons, but the key is to communicate desired behavior.

Stories describe how a product should work, so story development is a common place to include functional security requirements. The challenge is more *how* to communicate requirements than *when* in the process communication should take place. It must be simple so developers understand what they are being asked to do. Rather than a long list of requirements, try a simple state machine diagram to get the idea across. Clear examples of what you want and what to avoid are best. Simpler is usually better.

Product Management and Task Prioritization

The Product Manager or Product Owner assigns tasks. Their responsibility is to manage incoming features or stories, break work down into manageable tasks, and prioritize them. They often have

the least security knowledge on the team. The issue is deciding how to *prioritize* and *track* security issues amongst all competing priorities.

We recommend working with product management to assign tags or labels to security tasks and vulnerabilities, so they can be tracked like any other feature or bug. Balance security against other development tasks. Product management has a tendency to starve security tasks off the queue in favor of new features, while security people must resist labeling all their work as critical. If you integrate security into user stories, agree on reasonable priorities for different types of security bugs, and have tools to track security work, your working relationships with product owners will improve.

Development and Debugging

Development teams have not always been tasked with debugging their own code. In the Waterfall days Quality Assurance teams were often handed *completely untested* code. One key problem Agile avoids is developers 'dumping' code, by requiring verification that code performs its basic function *before* QA accepts it. Many organizations now require developers to write test cases to verify that submitted code accomplishes its required functions. Some Agile teams take this especially seriously, assigning task cards to write tests *before* assigning the task to code a new feature. Speed and efficiency are key requirements, so these tests become part of the automated build process — which automatically rejects failing code.

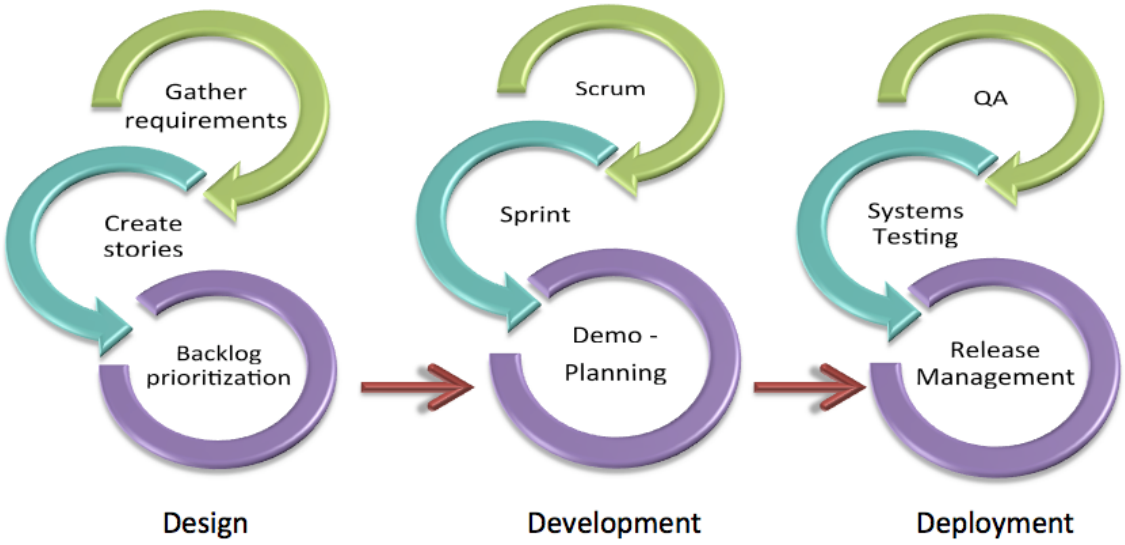
We recommend making security requirements and tests part of the story, but selecting small items that quickly help identify whether code passes or fails to meet requirements. You will not get complete testing in this phase, so don't bother asking. For new features you want the developer to understand the core security requirement, so have them construct acceptance tests to get an early idea of whether or not they are on track. For vulnerabilities and flaws ask for security regression tests to be included with the fix to *ensure* that mistake is not repeated. Stick to one or two simple tests to validate the specific need and leave the rest for later. If your organization automates static testing during the development phase you will need to integrate results into tracking tools so developers can leverage them while the code changes are fresh in their minds.

Quality Assurance and Release Management

Most organizations prefer to leave the bulk of testing until Quality Assurances gets their hands on the code to perform complete systems tests. That means penetration testing, dynamic app scans, platform compatibility, and integration testing all happen alongside functional and usability testing. But there is a problem. Have we mentioned that Agile with Scrum *does not have* a QA phase?

Instead development checks in small pieces of code and demonstrates functionality to the team — much more testing occurs earlier in the process. The good news is that you will be moving more security tests than ever before into automated development tests. The bad news is that the need for testing does not simply go away with a move to Agile. Internal security requirements or contractual agreements may require you to perform specific tests prior to shipping code, so you need to figure out when and where you can fit them.

Here is an example of a hybrid process with specific design, development, and QA efforts run in parallel, each as a distinct Agile process:



Some organizations run multiple Agile processes in parallel, essentially wrapping individual Agile processes into an Agile/Waterfall hybrid as in the above example. Some firms run only development Agile, with quality assurance and deployment running Waterfall. Which parts of your process are Agile are up to you — adjust your process to suit your needs, and the skills of your team members.

Your organization has likely come to grips with these issues already, performing final acceptance testing either as part of the normal sprint processes or as its own process. Our recommendation is to automate as much testing as possible into the build process, so issues are closely tied to the most recent changes and developers have an easier time understanding what caused problems.

There is no single Agile process. Agile is what you make it! The process you build today will need changes over time to accommodate new people, tools, and requirements.

Your goal is to maintain the Agile pace of development, with security folded in. That means introducing security in ways that accommodate simple and efficient processes. In many cases tradeoffs must be made, so in true Agile style, try it for a few cycles and change and improve over time. Agile has proven that when you break things down to simple components, you gain new and unique options to address development challenges — and this works for security too. You will find alternatives that are sufficient to get security into your products, and may find ways to do security testing and integration *better* than before.

Building a Security Tool Chain

Now that we have laid out the Agile process, it is time to discuss where different types of security testing fit within the process. Your challenge is *not* just to figure out what testing you need to identify code issues, but also to smoothly fit tests into the framework to speed testing. You will incorporate multiple testing techniques into the process, with each tool or technique focused on finding slightly different issues. Developers are clever, so development teams find ways to circumvent security testing if it interferes with efficient coding. And you will need to accept that some tests simply do not fit in certain parts of the process, while others can be incorporated in multiple places. To help you evaluate both *which* tools to consider and *how* to incorporate them, we offer several recommendations for designing a security *tool chain*.

Pre-Sprint Tests and Analysis

- **Threat modeling:** Modeling is looking for design-level security problems from the perspective of an attacker, and then designing countermeasures. The process enables designers and developers to think about the big picture security of an application or function and then build in defenses, rather than merely focusing on bugs. Classic vectors include unauthorized escalation of user credentials, information disclosure, repudiation (*i.e.,* injecting false data into logs), tampering, spoofing, and denial of service. For each new feature all these subversion techniques are evaluated against every place one user or code module communicates with another. If issues are identified the design is augmented to address the problem. In Agile these changes are incorporated into user stories before task cards are doled out.

- **Security defect list:** Security defect tracking covers both collecting security defect data and getting the subset of information developers need to address problems. Most organizations funnel all discovered defects into a bug tracking system. Defects may be found in normal testing or through any of the security tests described below. Security testing tools feed defect tracking systems so issues can be tracked, but that does not mean they provide consistent levels of information. Nor do they consistently assess criticality. How you integrate and tailor defect feeds from test tools, and normalize those results, are important for effective Agile integration. You need to reach agreement with the product owner on which defects will be addressed during each sprint, which will be allowed to slide, and how long. The security defect backlog should be reviewed each sprint.

- **Patching and configuration management:** Most software uses a combination of open source and commercial code to supplement what the in-house development team builds. For example

Apache supports most current web services. Keeping these supplementary components patched is just as necessary as fixing issues in your own code. Agile offers a convenient way to integrate patching and configuration changes at the beginning of each sprint: catalog security patches in supporting commercial and open source platforms and incorporate those changes into the sprint as tasks. This pre-supposes IT and its production systems are as Agile as development systems, which is regrettably not always the case.

- **Metrics and Policy Management:** It's important to track development tasks, features, open issues and vulnerabilities. Having tools that provide a view into what's happening on a daily or weekly basis is important to maintain control and efficiency of the development process and identity bottlenecks that slow the process down. Similarly, being able to view policies and metrics across the entire development organization, and not just a single development team, provides a picture of how teams use code, how subtle differences in process effect their productivity, and help keep the organization at large on track. We recommend taking the time to set up these tools in advance, at the very least to capture metrics, so you can measure performance over time.

Daily Tests

- **Unit testing:** Development teams use unit tests to prove delivered code functions as designed. They are typically created during development; teams using test-driven or behavior-driven development write them *before* the code to be tested. Unit tests run after each successful build and help catch defects caused by recent changes. Unit tests often include attacks and garbage input to verify that the application is resilient to potential issues outlined during threat modeling. The formal requirement for this type of testing needs to be included in Agile user stories or tasks — they will not magically appear if you fail to specify them as a requirement.

- **Security regression tests:** Regression tests verify that a code change actually fixed a defect. Like unit tests they run after each successful build. But unlike unit tests each regression test targets a known defect — either in the code under development or in a supporting module. It is common for development teams to break previous security fixes — particularly when merging code branches — so security regression tests are an important failsafe. With each security task to fix a defect, include a simple test to ensure it stays fixed.

- **Manual code inspection:** Code reviews, also called peer review, are where one member of the development team examines another developer's code. They ensure code complies with general standards but also look for specific implementation flaws such as unfiltered input variables, insufficient user authentication, and unhandled errors. Despite wide adoption of automated testing, about 50% of development shops still leverage manual code review to assess code quality. Manual review may appear inefficient, but properly focused it can be very Agile. The team may choose to perform these tests during development, QA, pre-deployment, or any combination, depending on the Agile flavor in use. This task can be assigned to any developer, on any branch of code, and as focused or random as the team decides. We recommend manual testing *against critical code*

modules, such as authentication, encryption, session management, and other sensitive code pieces. Supplement inspection with automated code scanning because manual scanning is repetitive and prone to errors, but catches errors which automatic scanners are prone to missing (especially logic flaws).

Every-Sprint (Commit) Tests

- **Static analysis:** This refers to examination of the source code of a web application for common vulnerabilities, errors, and omissions within the constructs of the language itself. This provides an automated counterpart to peer review. Among other things these tools generally scan for unhandled error conditions, unfiltered input variables, cross-site scripting, cross site request forgery, command injection, object availability and scoping, and potential buffer overflows. The technique is called "static analysis" because it examines static source code or binaries, rather than execution flow of a running program.

Like manual reviews, static analysis is effective at discovering 'wetware' problems: issues in code directly attributable to programmer error. Better tools integrate well with various development environments to provide educational feedback and suggest corrective actions; prioritize discovered vulnerabilities based on included or user-provided criteria; and include robust reporting to keep management informed, track trends, and engage the security team in the development process without requiring them to be programmers. Static analysis cannot examine all code pathways, and it is blind to certain types of vulnerabilities and problems that only manifest at runtime. To address these gaps dynamic analysis tools (below) often supplement static analysis.

Some static analysis tools require all code to be present and compiled before they can be used, while others can work with isolated components. This trait dictates whether static analysis can run each build, or must wait until the end of the sprint. Organizations can opt for analysis inside their environment, or send it off to a cloud service. On-premises static analysis tools give developers more local control over scanning runs and schedules, additional flexibility with custom rules, and do not require code to leave your environment — the latter a critical factor for many organizations.

In comparison, cloud-based services require fewer in-house personnel to manage, are more scalable, and ensure the consistent application of security policies across all development teams. Some cloud services also address the risk of source code exposure through binary static analysis, meaning they examine compiled code and do not require access to source code.

Every team is different but many run static analysis as part of the nightly build process, only responding immediately to critical vulnerabilities. Some teams also opt for more-frequent sandbox scans, which are run before moving to the main repository. They generally also run static analysis at the end of each sprint, agreeing on which issues must be addressed before code changes can be bundled into the final build.

- **Dynamic analysis:** Dynamic analysis covers gaps left by static analysis: it identifies problems which cannot be detected in source code, as well as issues which are more visible during

execution. Dynamic analysis tools fall into two types: white and black box. The test application that 'exercises' the application may lack all prior knowledge — probing the application as a "black box" and looking for exploits as it navigates the application. More often tests are a "white box" derivative of functional tests, traversing known pathways and supplying malicious or garbage input values. Random black box testing represents an outside hacker without detailed internal knowledge and unburdened by assumptions, but takes longer to run and cannot focus on known or suspected weak spots. White box tests can prioritize known critical code paths and focus on high-priority issues.

- **Component analysis:** Most applications are deployed in ecosystems, rather than tiny isolated boxes — and containers aren't really isolated. They rely on third-party code libraries, operating system elements, and a slew of other open source and proprietary tools and components. Any of these can introduce vulnerabilities just as easily as internal development error. Component analysis provides security testing for those pieces, ensuring that expected components (such as the proper compiler version or run-time engine) are in use.

It is important to test both credentialed and anonymous access. Some vulnerabilities may not be visible to random attackers, but might show up when logged in as a known user. With both white and black box testing human analysis of results is necessary to distinguish real problems from false positives. Error conditions are easy to find, and most dynamic testing tools discover and report many, but behavioral oddities which indicate attack vectors are not usually detected by scanning alone. Some forms of scanning are run during daily automated build testing, but more often they require tests to be updated after code delivery, and are often performed after specific functional milestones. Dynamic and static analyses are complimentary, with different strengths, for slightly different audiences. Some vendors provide both, combining results and reports.

Additional Pre-deployment Tests

- **Vulnerability assessment:** It is extremely useful to assess applications for anything an attacker could potentially use. Application-layer vulnerability assessments are very different than general vulnerability assessments, which focus on networks and hosts. Application-layer assessments validate configurations, detect known defective code modules, check for patches, and evaluate user authentication services. Some dig a bit deeper to examine application structures, and perhaps metadata and logic; they might even use customized assessments (*exploit code)* to determine whether applications are vulnerable.

- **Penetration testing:** The goal of a vulnerability assessment is to find avenues an attacker can exploit; a penetration test goes a step further to validate which attack pathways pose actual risk. A penetration tester attempts to exploit applications to determine what a real attacker could do, and what the consequences might be. Pen tests are important because they closely replicate the techniques and tools attackers use to compromise applications, but we find structured — as opposed to ad-hoc or piecemeal — penetration tests more valuable for risk prioritization. Penetration tests can assess the entire vulnerability stack from the network and operating system

up through custom application code and application firewall layers. To examine the entire stack, tests must be run against live production services.

The earlier a defect is discovered, the easier and cheaper it is to fix. The current trend is to shift all testing, including security, as early in the process as possible. Building up a suite of nightly automated tests is an increasingly common strategy to improve quality and security.

Integrating Security with Deployment Pipelines and DevOps

Now it is time to discuss directly integrating security with Agile. Agile itself is in the middle of a major disruptive evolution, transforming into a new variant called DevOps, bringing significant long-term implications which are *beneficial to security*. The evolution of development security and Agile are closely tied together, so we can start by specifying how to integrate into the deployment pipeline, then discuss the implications of DevOps.

Understanding the Deployment Pipeline

The best way to integrate security testing into development is by integrating with the *deployment pipeline*. This is the series of tools an organization uses to take developed code from the brain of a developer into the hands of a customer.

- **Integrated Development Environment (IDE):** The IDE is where developers write code. It typically consists of a source code (text) editor, a compiler or an interpreter, a debugger, and other tools to help programmers write code and build applications.

- **Issue Tracker:** A tracker is basically a project management tool designed to integrate directly into the development process. In Agile development user stories are entered directly, broken down into features, and broken down again into specific developer tasks/assignments. Detected bugs also go into the issue tracker. This is the central tool for tracking the status of the development project — from earliest concepts to updates to production bugs.

- **Version Control System/Source Code Management:** Managing constantly changing code for even a small application is challenging. Source code is mostly a *lot* of text files, which may be worked on by teams of tens, hundreds, or thousands of developers. The version control system/source code management tool keeps track of all changes and handles checkout, checkin, branching, forking, and otherwise keeping the code consistent and manageable. Whichever tool is used, the collection of code it manages is typically referred to as a *source code repository,* or *repo* for short.

- **Build Automation:** Automation tools convert the text of source code into compiled applications. Most modern applications include many components which need to be compiled,

integrated, and linked in the correct order. A build automation tool handles both simple and complex scenarios, defined by scripts created by developers.

- **Continuous Integration (CI) Server:** Continuous Integration is the next step of build automation. It connects to the source code repository and uses rules to automatically integrate and compile code updates as they are committed. Rather than a manually invoked build automation tool, a CI server grabs code, creates a build, and runs automated testing automatically when triggered — such as when a developer commits code from an IDE. CI servers can also automate the deployment process, pushing updated code onto production systems.

There are an unlimited range of possible deployment pipelines, and a pipeline is often actually a series of manual processes. But the broad steps are:

1. The product owner enters requirements for a feature into the issue tracker.

2. The product owner or someone else on the development team (such as the program manager) breaks the user story and features down into a set of developer assignments, which are then added to the Backlog.

3. The program manager assigns specific tasks to developers.

4. A developer checks out the latest code, writes/edits in an IDE, tests and debugs locally, and then commits to the source code repository using the version control system.

5. The build automation tool compiles the code into the application and may perform automated testing. The compiled product is then sent to QA/testing, and eventually to Operations to push into production. If something breaks that is logged as a bug in the issue tracker, and the process can't proceed until it is resolved.

6. If the organization uses continuous integration the code will be automatically compiled, integrated, and **tested** using the CI server. It may be pushed into deployment or handed off for additional manual testing, such as user acceptance testing. Again, if something breaks that becomes a bug in the issue tracker, probably automatically.

Not every organization follows even this general process, but just about everyone running Agile uses some variation of it.

Integrating Security

If you map our security toolchain to the deployment pipeline there are clear opportunities for integration. The ones we most commonly see are:

- Security manages security issues and bugs in the issue tracker. Security features are often entered as user stories or feature requirements, in coordination with the product owner or program manager. Security sensitive bugs are tagged as security issues. In some cases security

teams monitor the issue tracker to help identify potential vulnerabilities that were entered as non-sensitive normal bug reports.

- Static analysis is integrated in the IDE, build automation tool, or CI server. Sometimes all of the above. For example when a developer commits code locally it can undergo static analysis, with issues highlighted back in the IDE for easy identification and remediation. Static analysis may also be triggered when code is committed to the source code repository.
- Dynamic analysis is also typically integrated at the build automation or CI server, using tests defined by security.
- Other security tests, such as unit, component, and regression testing, are often best integrated in the build or CI server.
- Vulnerability analysis may be automated if the organization uses a CI server, but otherwise is often manual or periodic.
- Any problems discovered by testing tools generate entries in the issue tracker, just like any other bugs. Ideally security must sign off on any unremediated security bugs before release.

Security and DevOps

There is no single definition of DevOps, but essentially it means deeper integration of Development and Operations in the software deployment process. DevOps *integrates* more Operations in Development, and more Development in Operations. You may remember corny ads for Reese's Peanut Butter Cups: "You got your peanut butter in my chocolate! No, you got your chocolate in my peanut butter!" DevOps is a bit like that. Integrating the produces valuable synergies.

Putting code into production is a *very* important step, one we haven't fully addressed in this series because it is generally not part of the Agile development discussion. This step is incredibly error-prone because each developer tend to work in a personal vacuum, separated from production systems and the operations folks who deploy. That separation is often a security and compliance *requirement*. But as a result development and testing environments diverge widely from production, and the differences only grow over time. This materially increases the likelihood of things breaking when they are deployed because developers cannot account for all the differences.

The deployment process is also very manual, typically with Operations teams running through deployment steps to get everything configured correctly on production systems and then updating the code. Even if an organization builds code using Agile, they often deploy infrequently to avoid breaking production: Agile development but Waterfall deployment!

DevOps addresses this through extensive automation. Thanks to virtualization and cloud computing, we have tools to consistently and programmatically define our environments and "rebuild production" as we update code. This works thanks to two key pieces:

1. **Configuration Management:** Newer configuration management tools completely manage and define a system using programmatic scripts. In DevOps we use these tools to define and push server configurations to enforce consistency and update automatically.

2. **Cloud and Virtualization Environment Configuration Tools:** Some cloud providers support building out complete environments based on templates. There are also stand-alone tools to manage the same automation. At a basic level you can define networks, servers, configurations, and connections — all using scripts.

In both cases the tools are largely driven by text files from a version control repository, just like code. This enables us to completely define the *environment,* the *server configurations,* and the *application* in text files — and then to build, test, and deploy consistently.

This supports *continuous deployment,* the heart of DevOps. A developer working with Ops defines server and environment requirements within the development process. Both Development and Operations use the same deployment pipeline, which now updates not only code but also the environment in which it runs. This means, for example, that the CI server can create an entire test environment (on a local virtual platform or in the cloud) which is completely consistent with production. Integrating operations testing with development detects more problems earlier.

DevOps dramatically improves consistency and predictability, while (thanks to automation) also speeding up deployment if desired. It can be an extension to Agile, aligning with sprints, or become a truly continuous process where code is deployed into production as soon as it is complete and passes automated tests. Some organizations using DevOps deploy new code into production dozens of times a day without sacrificing security or resiliency. Errors still occur, but they are smaller, with a faster feedback cycle, and improved ability to roll back to known good states because *everything* is in version control.

DevOps is ideal for security integration because it enables us to integrate security requirements and testing into Development and Operations — often completely automated. This *continuous integration* is another key element of DevOps, ensuring all the code and supporting services work together, and providing an ideal location to *automatically* integrate security. For example we can add server hardening requirements into configuration management scripts. We also get improved audit trails because all production changes start at the beginning of the pipeline, rather than with administrators modifying servers in production.

This is only one aspect of DevOps, but it is the one most relevant to security teams and Agile development. DevOps itself is incredibly disruptive because it increases agility and resiliency while reducing operational overhead costs. At the same time improved consistency and better enforcement of security standards — without impeding the development process — bring tremendous security advantages.

Summary

Security's job is to manage risk for an organization. That is impossible without *understanding* the risk, and few things are harder to understand than untested new applications. Development teams and security have historically battled due to their competing priorities: Development needs to hit deadlines and get code out the door, while Security needs to understand the code and ensure it's safe.

If Security sits on one end of the pipeline and tries to evaluate and fix everything before it escapes through the factory door there is no way they can keep up. By the time they see problems — especially in a rapid Agile environment — it is too late in the process, more expensive to fix, and sure to hold up business needs.

> Agile, deployment pipelines, and even DevOps may look like they break security, but actually enhance it through standardization, faster feedback cycles, and automation.

The answer is to integrate as directly as possible into the development process. Start early with threat modeling, define security requirements, share an issue tracker, and automate security testing into the development (or deployment) pipeline. Agile is all about being more responsive and better meeting customer requirements, and these days nobody can credibly argue that security isn't a key requirement. You just need to understand where developers are coming from and how to integrate into *their* process — trying to make development processes conform to security has a very poor history and Agile only exacerbates the problem.

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.

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.

Rich Mogull, Analyst/CEO

Rich has twenty years experience in information security, physical security, and risk management. He specializes in cloud security, data security, emerging security technologies, and security management. Rich is the primary developer of the Cloud Security Alliance CCSK training program. Prior to founding Securosis, Rich was a Research Vice President at Gartner on the security team where he also served as research co-chair for the Gartner Security Summit. Prior to his seven years at Gartner, Rich worked as an independent consultant, web application developer, software development manager at the University of Colorado, and systems and network administrator. Rich is the Security Editor of TidBITS, on the advisory board of DevOps.com, and a frequent contributor to publications ranging from Information Security Magazine to Macworld. He is a frequent industry speaker at events including the RSA Security Conference, Black Hat, and DefCon, and has spoken on every continent except Antarctica (where he is happy to speak for free — assuming travel is covered).

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/>.