MODERN
CYBERSECURITY

CHAPTER 5

Hardening the Value Stream

by Bryan Finster

CHAPTER 5

Hardening the Value Stream

Bryan Finster

Introduction

Cybersecurity threats are always evolving. What was once the realm of individuals or gangs breaking into systems for fun or profit has become national actors waging information warfare. Exploiting simple vulnerabilities due to insecure coding, poorly secured environments, outdated policies, etc. has transformed into supply chain attacks where vulnerabilities are injected into "trustworthy" building blocks upstream of the systems we are building. We must evolve how we think about security to counter these threats. The industry is slowly evolving from legacy Software Development Life Cycle (SDLC) practices towards more effective value delivery methods.

In the SDLC process, we see the typical stage gates:

- Requirements
- Design
- Coding
- Testing
- Delivery
- Heroics
- Repairing customer relationships
- Resolving crushing defect loads
- Declare tech debt bankruptcy and start over

This has obvious disadvantages for customers and the organization. It delays feedback and limits quality inspection until development is complete.

Defining Quality

Quality is far more than "does it function as designed?" Quality means that it meets the needs of the users. It is stable, available, fit for purpose, and secure. Quality cannot be inspected into being, it is the outcome of building quality feedback into every process. To improve outcomes, we need to change how we think about quality. What does quality look like? How does it happen? We need to engage everyone in making quality better. We need to build a working environment and culture that improves the developer experience so that building functional, performant, stable, and secure systems is not a constant uphill battle.

There is more movement in the industry to improve outcomes by improving how value flows through organizations. Quality processes are evolving from an inspection step to continuous activity and a daily habit. This is a journey and not something that improves overnight. While many people new to DevOps understand it to mean "development and operations working towards common goals", in reality, DevOps is about improving that quality feedback. However, in the rush to get better at improving the functional, performance, and stability aspects of quality, security is often overlooked. It is still frequently seen as the role of specialists instead of yet another quality aspect that must be part of the flow. This has given rise to people using the term "DevSecOps" to remind people that security shouldn't be left behind. For security to be effective, we need the same mindset. How do we move security closer to the work and how do we enable teams by making security easy?

Legacy Approach to Security

The typical security process is still aligned to SDLC where security checks are designed to be done on the finished product. This becomes a handoff in the flow and is seen as "other than" development and the responsibility of the Security team. Since these inspections frequently require heavy manual effort, it's not uncommon to see these activities implemented as a one-time or infrequent "security review" that results in the application receiving a security certification to operate. These usually include policies for how often recertification is required and that recertification will be needed if major changes are made to the application. As teams move from project to product management and from big bang delivery to very frequent delivery of small changes, this security certification process becomes security theater.

"Security theater" is the act of going through the motions to "assure security" without meaningfully increasing how secure we are. When our process is designed for inspection at the end, it ignores the reality of how development actually occurs and of the constantly evolving threat profile. Something that is secure today can be exploited tomorrow and will need to be resecured. A dependency with no known vulnerabilities can be the victim of a supply chain attack from another upstream dependency. When we upgrade the dependency we are now vulnerable. Even if we don't upgrade a dependency, a vulnerability can be found later. The one-time certification process, no matter how extensive, does not mean it will be safe in the future.

We need continuous verification of security just as we do for every other quality aspect. To move from security theater to continuous security, we need to apply the same principles we do to other forms of continuous testing. What activities are rules based? Automate those. What activities require human creativity? Those need to be done continuously without blocking the flow of delivery. This is critical for improving quality. Any validation or compliance activity that is both manual and required before every delivery will increase the batch size of every delivery. Large batches of work hide bigger problems and delay value. This creates an incentive to not do these activities as frequently so that we can meet organizational delivery goals.

We can see this happening often with functional testing. If days or weeks of testing are required for every release, then testing is done on large changes except when there is an emergency and then the tests are skipped. For functional testing this can result in regression in one area while attempting to fix another area. If we have a security process that incentivizes the same behavior it's not bad, it's dangerous. We cannot afford to leave ourselves exposed by continuing to use methods that are impossible to execute on daily code changes. Trying to "inspect in security" after the fact is just as ineffective and more dangerous than doing the same with other quality aspects. Instead, the focus should be on how we build security into the flow of delivery.

Fear Driven Development

> "We'll make teams accountable to security. If they cause a breach, we identify the guilty developer and terminate them. Now we are DevSecOps!"
>
> ~Bryan Finster

In organizations with legacy testing and security practices, it's common to find legacy postmortem practices. "Document what happened and who caused it!" In 2017, Equifax experienced a data breach that exposed sensitive data for 145 million accounts. When former Equifax CEO Richard Smith was questioned by Congress, he said it was the fault of a single person. Smith testified, "Both the human deployment of the patch and the scanning deployment did not work. The protocol was followed. The human error was that the individual who's responsible for communicating in the organization to apply the patch, did not."

This, of course, is nonsense. The only time failure is a single person's responsibility is when only a single person is involved in delivering value. The failures were guaranteed by how they did their work, but this statement gives us an insight into their culture and why a breach was probably inevitable.

What is the insight we can get from this? Fear Driven Development is

the scapegoat culture of "if something bad happens, we will identify the person responsible and hold them accountable." This generates a culture of hiding problems either because we are afraid of being blamed for something we didn't do or we did do it but the consequences of trying to fix it are too high. Lack of trust in the value stream does not yield better value. Lack of trust means that if someone identifies a problem, the safest thing to do is ignore it and hope someone else gets blamed. This doesn't make us safer.

We need to create an environment of trust so that when failure happens, and it will, we can use postmortems to identify the failure in the system instead of finding a scapegoat.

PLATFORMS SHOULD ENABLE BETTER OUTCOMES

We need to make security validation more efficient, more effective, embedded throughout the entire process, and owned by everyone in the value stream. We need to make security part of the environment, culture, and daily work. A well designed platform can help.

What is a platform? A platform is a system created to enable others to solve problems with less friction. A delivery platform is a system that allows product teams to focus on what they are delivering and how to improve the delivered quality without having to build the basic infrastructure of delivering changes. A well-designed delivery platform needs to balance reducing friction and toil while ensuring organizational non-negotiables are met.

First, it needs to make it easier to deliver without expertise on how it works. This seems obvious, but many platforms require a steep learning curve and some teams may find it easier to use whatever poor tools they already know. There may be a temptation to create a "release management" team that is responsible for building delivery pipelines for the development teams, but this leads to disincentivizing quality ownership because it increases the difficulty of improving quality gates. To get the outcomes we want, the platform must focus on improving the developer experience to reduce this learning curve so that teams have total quality ownership.

Next, the platform needs to implement security and compliance rules to make sure that any non-negotiables cannot be forgotten. Having a common platform is a major win for security and compliance because Audit can have a central location to verify those are happening instead of requiring audits of every pipeline.

PLATFORMS REPLACE CAB

> "Change Advisory Boards (CAB) increase the cost of every change and encourage large delivery batches that are hard to verify. Because of this, they reduce the quality and increase the cost of every delivery. They are also ineffective at verifying if non-negotiables are being met because they are too far from the work. In one organization, our Compliance team was pushing for everyone to use CABs to ensure "two eyes on every change" to prevent bad actors. Bad actors are an important threat vector to check for, but there was no way for the CAB to verify no bad code was added to a release. Also, this requirement would halt our organizational continuous delivery goals, so we needed to find a platform solution. We asked Compliance, "Is the requirement that we have a change board or that every change is reviewed by more than one person?" They informed us that review was the requirement. We suggested that code review met that requirement and that our global platform could create a pipeline gate that would reject changes that did not have a code review. They were satisfied with that, dropped the requirement for CAB as long as they had audit ability for the gate, and we moved forward with our continuous delivery improvement goals."
>
> ~Bryan Finster

The platform controls must be balanced. It can enforce any policy that can be described in code but this is a double-edged sword. Implementing the right controls is important, but if we get carried away and create non-negotiables that are outside the bounds of security and compliance, then platform customers will be incentivized to find other solutions. We can

use the platform in opinionated ways to encourage broad improvements, but only if we do it in ways that make the right thing the easy thing to do. Making the right thing easy and the behaviors we want to improve harder, but not impossible, enables the organization to migrate to better patterns without removing the ability for older development efforts from using the platform at all.

> "As part of an effort to make sure that teams were testing code and writing clean code, the platform team decided to implement pipeline gates that would block any changes if there were code style issues or test coverage was below 80%. While the intent was good, "teams should use good coding practices", the result was the platform could no longer support anything except new development. Applications that had been written and supported for years could not use the platform without major refactoring to meet the platform team's opinions. For example, builds would break if conditions were nested too deeply or variables were named with patterns that did not meet the opinions of the platform team. Additionally, the mandated code coverage incentivized tests written for coverage instead of proper testing. These poor tests hide that behavior is not being properly validated. They are less valuable than not having those tests at all."
>
> ~Bryan Finster

A platform can be opinionated to act as a lever for improving how the organization delivers, but if it is too opinionated, it will generate a noisy quality signal that can reduce the effectiveness of our compliance and security goals by incentivizing workarounds. We cannot force testing or security with tools. We need to use the tools to make the right way the easiest way. We can make less desirable behaviors harder, but making them impossible will create perverse incentives.

A good platform must be obsessed with its customers, the development teams who need the platform to deliver better value. The platform team should not be attempting to keep the teams in line or force them to behave. They must act as partners who are helping their customers by

making safety, efficiency, and effectiveness easy.

It is important to understand that tools will never create quality though. Quality is always an iterative process of identifying and testing for known poor quality. This is a creative process of asking "what can go wrong?" and testing for that. This applies to security testing just as it does to functional testing. We cannot test for vulnerabilities we are unaware of. We can test for known vulnerabilities and use our creativity to predict others, but the platform alone will never make us safe. Platforms cannot enforce creativity, only repeatability. We need to design automation that can provide feedback of poor quality as close to the source of that poor quality as possible so that the teams can improve their outcomes through immediate feedback.

Federating Security

Security must be a partnership, not a dictatorship. Just like functional quality, security will not occur because we are told to be more secure or follow the rules. It cannot be created with dashboards or tools. Tools can enable it and data can alert us, but we need a culture of security to make us safe. This requires education, time, evangelism, and partnership. A security team should not be seen as the police who are, as one penetration tester once told me, "investigating the crime of new development."

Security is a key part of the value stream. Every part of the value stream needs to focus intensely on one thing: helping each other to optimize delivered value. If we use legacy security processes and increase the cost of every change above the value delivered, we are only creating waste. If we only use our security process for spot checks a few times a year, we are using security theater. We need to find a better way. At the same time as we work to federate testing by making it part of development, we need the same pattern for security. No one wishes to deliver insecure solutions, but the gap between what teams need to know and what most teams know is usually large. Security can be a service provider by aligning to specific value streams, partnering with development teams in that value stream, and learning the specific challenges they have in their context.

WHAT DOES SECURITY AS A SERVICE LOOK LIKE?

> "Our company hired a new security architect and aligned him to a development area where he could see the problems developers were having. He saw constant problems with development machine specifications and system access that were impeding their ability to deliver. He developed a new plan for provisioning and network access, worked with the security team and CTO to get buy-in, and helped the teams solve this recurring constraint in the value stream."
>
> ~Bryan Finster

By becoming familiar with the teams and their specific delivery contexts, security solutions can be less generalized, and more trust can be established between Security and Development. Constant communication will also help with the continuous education that teams need to keep current with the threats in their environment. In addition, the teams can help improve the tools to continuously improve the efficiency and effectiveness of automated scans.

Security Hobbyists

We want to push testing and security further left in the value stream and we want them embedded into everything we do. We have a problem though. This is a new way of working. We shouldn't simply direct it to be so and expect it to happen, yet that happens all too frequently. When bad outcomes occur, the response is "developers don't care about testing" and "developers have no interest in security". Is that really true?

Who decides what developers care about? The leadership in the organization. Everyone cares about what they are incentivized to care about. Incentives can be intrinsic, but if the organization has operated with testing and security as "not development", even those who care can get discouraged and fall into line with what the leadership rewards people for. Now we want them to care and we've changed the incentives, but

where is the support they need? We are depending on them to be good at it, but are we investing in that happening or depending on them to just pick it up on the job? On-the-job training is fine as long as the majority of the organization isn't doing that together at the same time.

> "I spent years as a developer before I ever saw a testing framework. Even then, management did not value testing. "We are falling behind on our deliverables. We can worry about testing later!" was an all too common response to the perceived slowdown from writing "extra code". I was never taught effective testing techniques. Instead, I found myself being "quality curious". Learning to test became my hobby because I wanted to understand how to implement a continuous delivery workflow. I knew that automated validation of delivery fitness was core to CD, but I had no real experience beyond a few simple tutorials on unit testing and some coding exercises using test-driven development. I was left to sort out the good and bad information about testing on my own. I even fell into the trap of thinking that 100% test coverage meant that an application was well tested. It took years of trying and failing to become competent. I look around and all of the advances I've seen from other developers in testing have been from the same hobbyist process. I wonder how much further we could have come and how much better the company outcomes could have been if our company had invested in growing our knowledge with dedicated training from reliable sources instead of expecting us to pick it up at home after hours?"
>
> ~Bryan Finster

Stories like this are the norm, not the exception. Even Google went through a phase of ignoring this problem until they finally assembled the "Test Mercenaries" and the "Test Certified" program to systematically upskill their development teams instead of hoping hobbyists learned the right things and helped spread good practices.

This same problem occurs with security. Developers are generally expected to be security hobbyists. Guided, self-paced learning can be

effective, but self-guided learning paths are seldom effective. Which information is good or bad? Which is outdated or doesn't apply to the current problem? Functional and performance testing are solved problems. The patterns for effective testing are well understood. While it requires good resources to understand the best practices, the practices themselves have been tested over time and only the tools see significant changes. Security is a whole other ballgame.

Security threats are constantly evolving and even good practices today can cease to be good practice tomorrow. Aligning everyone to this changing environment means depending on hobbyists and centralized security organizations is risky. We shouldn't put the safety of our organization at risk by ignoring the need for dedicated training. How can teams help protect the organization from attacks if they are not trained in how attacks are executed? Telling teams to follow rules isn't enough. People need to understand why vulnerabilities exist. Where do most attacks come from? Would it surprise them to know they are internal? What motivates attackers? All of these things are important for teams to understand so that they not only have the right knowledge, they have the right mindset.

Modern development is a complex activity and organizational success depends on investing in improving the value stream by investing in the people who make value flow. We cannot survive on hope, security hobbyists, and "someday, I'll read that book."

Grow a Culture of Security

Ultimately, security—just like every other aspect of quality—is everyone's business. If we want more secure outcomes for our consumers, we all need to live and breathe security. It cannot be the responsibility of a single group. Instead, it must be a cultural habit. We need to reduce the toil of staying secure. We need the right tools, implemented in the right way, that amplify security feedback. Quality is a function of how quickly we are aware of poor quality. Security is a dimension of quality; security is not different from functionality, performance, or stability in the need for rapid feedback.

The one component where security is different from the other quality aspects is the speed of change. Functional quality is a well-understood problem. While not everyone may be educated in it, because of the trend towards depending on hobbyists, the problem itself is just as well understood as the game of chess. Security threats, on the other hand, are constantly evolving and growing. With the growth of state actors, there are more resources than ever directed at finding vulnerabilities. Ransomware attacks keep criminals coming back because they are lucrative. How are you preparing your organization to work together to head off the next attack, while improving outcomes for your consumers, your bottom line, and everyone in the organization?

ABOUT BRYAN FINSTER

I've been a practitioner of continuous delivery and the DevOps principles that support it for many years. I'm a pragmatic developer who focuses on the goal of how we improve the continuous delivery of value to the end user every day. I will do what it takes to remove waste, improve bottlenecks, and deliver stable, available, secure, and useful solutions that can be rapidly evolved without heroics to improve the lives of customers and developers.

Deploy more. Sleep better.

Bryan Finster's LinkedIn