Contrast SECURITY

EBOOK

Application security: Five principles for 'Shifting smart'

Introduction

Many in the application security community have been obsessed with “shifting left”—that is, moving application security testing earlier in the software development life cycle (SDLC).

Shifting left was a useful concept a decade or two ago when security testing was not routinely done until late in the process. But recently, some organizations have been fixated on shifting further and further left beyond where it can be effective for many common security vulnerabilities.

Organizations need to take a step back and think about what makes the most sense in their specific context. In my opinion, rather than shifting left for the sake of shifting left, organizations should shift smart—optimizing security testing throughout the SDLC based on each application’s specific needs.

Here are five principles for shifting smart with application security.

Jeff Williams
CTO and cofounder of Contrast Security —helping companies become truly great at securing their apps and APIs.

1 | Harden your software stack.

You wouldn’t think of deploying a host without hardening it against attacks. But most organizations don’t harden the software stack that they run on those platforms.

Hardening your software stack with runtime protection will prevent vulnerabilities from being exploited—even if developers make mistakes.

This is a strong mitigating protection for most vulnerabilities, including published common vulnerabilities and exposures (CVEs) in third-party libraries, problems in custom code and zero-day vulnerabilities that were unknown before.

Hardening your stack can be done before any code even gets written and will allow you great flexibility in how you perform security testing and respond to new vulnerabilities.

2 | Test what matters when it matters!

Rather than blindly casting a net for everything in every application, you should prioritize security testing based on your threat model. OWASP, NIST and PCI software security standards all now require threat modeling.

You only need to test your defenses for the threats that you actually face. Standards are great, but be sure to tailor them to your business.

The types and timing of needed security testing can differ from application to application.

Fortunately, most of the development pipeline is automated these days, and the time from integrated development environment (IDE) to production is measured in minutes. So shifting some tests "right" to take advantage of thecontext available in a fully assembled and running application doesn't mean you have to give up the benefits of near real-time feedback to developers.

For example, you can eliminate SQL injection testing if your application doesn't have an SQL database. And you can test your authentication and access control mechanisms when they're fully deployed and configured.

SPOTLIGHT

Shifting security testing into the automated build pipeline, or into the integrated development environment (IDE)?

It is worth considering whether it's worth pushing security testing all the way left into the IDE—even before the software is compiled and packaged.

You lose a lot of context when you're only looking at source code.

There are many kinds of vulnerabilities, and it is naïve to think that shifting as far left as possible works for all of them.

Here are a few examples:

- **SQL injection.**
  The best way to do the data flow analysis necessary to detect these vulnerabilities is to observe actual data flowing through a running application. For this and other injection vulnerabilities, an organization is actually better off shifting a bit to the right and using interactive application security testing (IAST) in a test environment with the fully assembled application.
- **Authentication and authorization.**
  These vulnerabilities are critical and almost always in custom code that's unique to a particular application, making it extremely hard to find them in the early stages of development. Most organizations test for this with manual penetration testing very late in the SDLC—another shift to the right.
- **Weak encryption algorithm.**
  In most cases, this vulnerability doesn't involve complex data or control flow and can be found without a lot of other context. It is generally fast, easy and fairly accurate to search the use of weak algorithms in the IDE or code repository.

3 | Test with the best.

Builders cannot build a house with just a screwdriver, and they don’t put screws in with a saw.

Your goal should be to use the best testing technique for each of your defense strategies. You do not need to test every defense with every single technique.

Rather than trying to use every kind of tool on every kind of vulnerability, organizations need to select security testing approaches that deliver the optimal balance of fast, complete, accurate, easy and cost-efficient.

Seek out tools that provide strong evidence of coverage and accuracy for a class of vulnerabilities. Running weaker tools for that same type of issue is unlikely to make your overall results stronger and introduces opportunity costs for your teams.

For example, injection vulnerabilities are difficult to test with just source code. Interactive tools that trace real data through running code are far faster and more accurate. Authentication and access control are often custom-built and must be analyzed manually with code review or penetration testing.

4 | “Notify left.”

Even in cases when security testing shifts later in the process, notification should go left.

You should focus on how quickly the security feedback gets to those who need it and route it through the tools they’re already using.

While application security dashboards can be useful for managers, you don’t want developers having to log in and check a separate system. If you have fast and accurate vulnerability data, developers should see that information immediately so they can fix it as part of their normal work.

If you put vulnerabilities into a defect database, the odds are that they will never be fixed. Once vulnerabilities become part of the backlog, you’ll have to rely on expensive risk prioritization processes, selecting issues for sprints, service-level agreements, work tracking, retesting and so forth—all of which can be slow and expensive. A high-functioning organization may be able to remediate vulnerabilities within days. However, research has found that it typically takes months[1] for organizations to fix flaws that were discovered by static methods.

If information about a vulnerability gets back to the developer that introduced it quickly, that section of code is fresher in their memory—making the fix faster and easier.

[1] https://www.veracode.com/sites/default/files/pdf/resources/infosheets/soss12-healthcare-infosheet.pdf

5 | Optimize for learning.

You certainly should invest in regular security training for developers, as the application security landscape is evolving at least as quickly as the world of DevOps.

But bear in mind that developers learn a lot more by working on their own code than they do by sitting in a generic training session.

That is another reason that getting rapid feedback back to developers is critical—it can help them learn from their coding mistakes and not repeat them. Repeat that process a dozen times for different kinds of vulnerabilities, and you have a developer who introduces far fewer of them.

Ultimately, the most cost-effective application security program goes beyond finding and fixing vulnerabilities quickly. It helps prevent vulnerabilities from existing in the first place.

Summary

Many companies that try "shift-left" and "developer-first" security tools can experience an avalanche of false positives from code repositories that aren't in use, code that never runs, tools without enough context and tools that sacrifice accuracy for speed.

We must recognize that application security is a complex problem and rather than blindly shifting left or blindly shifting everywhere, organizations should *shift smart*. One key factor is to perform security testing only when you have enough "context"—**the details of how an application or API actually functions**—to accurately identify real, exploitable vulnerabilities.

If you want to learn more, or see how Contrast tests and secures applications visit **www.contrastsecurity.com** and click on the "Get DEMO" button on the top of every webpage to schedule a personal introduction to Contrast..

6800 Koll Center Parkway
Suite 235
Pleasanton, CA 94566
Phone: 888.371.1333

Contrast SECURITY

contrastsecurity.com