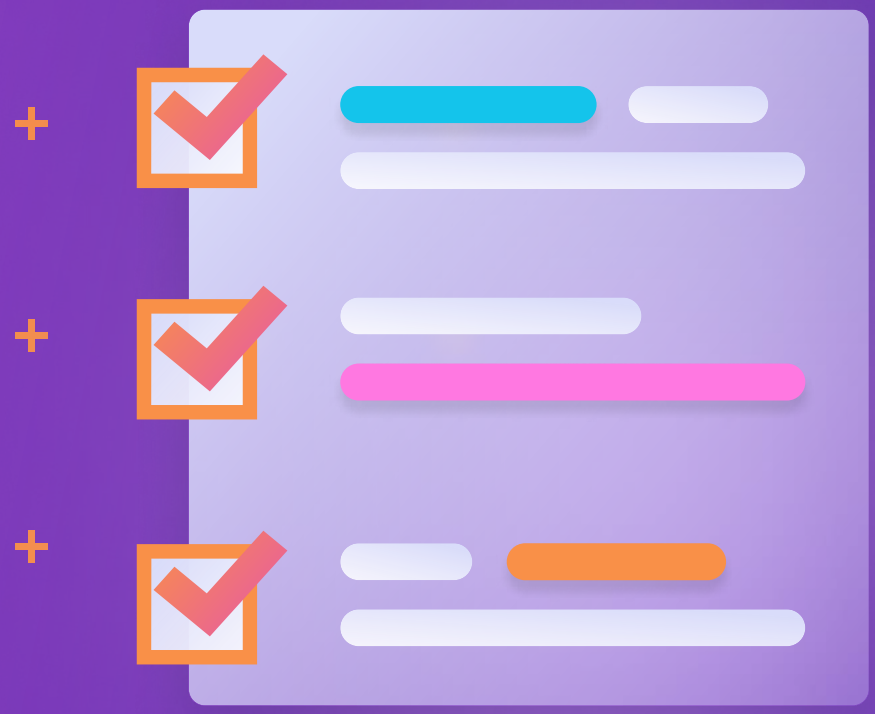

WHITE PAPER

CISOs Guide to Cultivating Developer Security

Author: Simon Maple, Field CTO, Snyk

snyk

Table of Contents

Understanding development teams

How empathetic and understanding are you to your development teams?

Are your developers empowered?

What drives devs to spend time on security?

Understanding the developer attitude to security

They do it because they should

How to improve security adoption in your dev teams

Techniques to improve adoption

snyk

Best Practices for Driving Developer Adoption of Security Tools and Processes

It is broadly accepted that the way to scale application security in modern organizations is to embed it directly into the development workflows as a natural part of the agile process and DevOps pipeline. This is much easier said than done, though, and it brings many challenges in terms of **acceptance** and **adoption** from developers and development teams. This white paper summarizes our learnings from discussions with a variety of organizations that have DevSecOps programs at varying stages of maturity. The common theme among all the individuals and groups we talked about was their want for development teams to adopt secure development practices as part of their workflows.

It's important to note that no two groups we spoke with do things the same way. It's equally important to say that what works well in one organization may well not work as well in another, since there are so many different factors which could influence the success of these practices. It's key for you, the reader, to identify practices and approaches within this paper that you feel would work best for your organization, culture, and teams.

snyk Cheat Sheet: Developer Security

People

1. Be empathetic to your development teams

- Understand their general appetite for change.
- Learn which teams and projects are likely to implement modern technology and development processes, as well as which are resistant to change.
- Determine how well project ownership and responsibility is defined.
- Understand team capacity for delivery, as well as other tasks, such as performance, reliability etc.

2. Embrace top-down business security prioritization

- Make it clear when prioritization comes from the business.
- Security teams must help development teams address the business's security prioritization, rather than adding more to their plate.

3. Enable developer-security collaboration

- Consider a security champions program or a security guild to make it easy to collaborate.
- Work with developers to build trust, rather than relying on fear to get things done.
- Build shared programs and have common goals that security and development work towards.
- Identify ownership in development teams so that security teams can be relevant with requests.

4. Educate to enable secure development

- Educate teams before rolling out programs.
- Recognize that certain education isn't relevant to all teams.
- Train teams on the specific vulnerabilities that affect them using previous incidents as examples.
- Track education via scorecards.
- Gamify education when possible.

5. Rewards and recognition

- Call out individuals for their achievements and security improvements.
- Highlight team successes, and program adoptions to show their progression and encourage other rollouts.
- Make sure the engineering leadership team is giving recognition and showing security is important to the teams
- Offer swag, gifts, travel to security conferences, etc. to help build awareness.

Process

1. Include developers in process-changing decisions

- Understand how development teams are integrating security practices into their pipeline today, and try to match their workflows.
- When new security rules or processes are made, work with development teams to set thresholds and working practices.
- Make sure education and awareness is shared around any process change — well in advance — including blogs, documentation, and learning hubs.

2. Make the right path the easy path

- Don't overcomplicate processes. Work with the development team to keep it simple.
- Make sure you have good documentation of how other teams work and how activities should be done.
- Provide paved road options for developers to be able to make the right decision by default.
- Integrate processes into their existing workflow rather than trying to create new workflows.
- Make sure processes (and tools) focus on solutions, not problems.

3. Create visibility and transparency across teams

- Scorecards are a great way to report progress and status of your security to the organization.
- Teams should be accountable to the scorecard results and prioritize security based on their health and their plans.
- Lean on developer values and gamification to bring work into sprints, using scorecard data as justification.

4. Speed of rollout

- Identify teams that are more adaptable and open to introducing new security programs.
- Roll out to those teams first, initially as a low touch, visibility, approach.
- Add secure development activities to development workflows focusing on the security of their incremental application changes first.
- Ramp up guardrails at the speed that the teams can consume change without overwhelming them.
- Identify more teams to roll out to after the early-adoption teams have shown value.
- Use best practices identified during the initial rollout to drive adoption.

Tools

1. Embrace developer tooling

- Developer security tools need great self-serve usage, including easy onboarding, intuitive workflows, and clear documentation.
- Tools must integrate into existing workflows, including IDE, Git, and pipelines.
- Tools must have rich APIs and be automation-friendly.
- Look for tools with wide adoption by the open source and the security communities.
- Try to pick a tool that covers the whole application, including first-party code, third-party libraries, containers, IaC, etc.
- Tools should focus on fix and remediation over find and report.

2. Don't overwhelm developers

- When introducing tooling to an organization or team, use it for visibility initially.
- Over time, dial up the rules and guardrails through policies to improve the secure development process.
- Back up any tooling changes with awareness, communication, reasoning, good process documentation, and education.
- Focus tooling on new code delivery first, followed by backlog issues to reduce load on developers.

3. Automate everything

- Integration into a workflow needs to be automated and done in the right place based on the team.
- As well as the testing in the workflow, look at other hooks where tools can be connected (ticketing, reporting, alerting, etc.).
- Be able to pull reporting data from your tooling, preferably from an API to populate your existing dashboards and scorecards.

Understanding development teams

Implementing any cross-functional program requires empathy. Collaborations between security and development teams commonly break down due to perspectives and goals being so different. While both teams may want the same end-result — secure applications — the preferred workflows and measurements of success can be very different. Fortunately, **empathy can drive alignment**.

When rolling out a modern AppSec program, it's crucial for those who interact and support the development teams to have a clear understanding of the problems, frictions, and frustrations the developers currently face. Only once they understand those, can they see how to integrate security into the development process. Let's dig deeper into the areas that are worth assessing and being conscious of before starting new collaborations. Below are some questions you can ask yourself before starting a rollout.

> *[Security] has got to be a partnership. A fundamental reason that a lot of our security fails occur is that we try and dictate to, rather than truly partnering with, the groups that we're trying to help be successful. My role is to help our engineering teams at Mandiant produce good, secure, functional code quickly. We need to move fast as a business, partnering is how we produce things that solve the business needs and security needs."*
> — Tim Crothers, SVP, Chief Security Officer at Mandiant

How empathetic and understanding are you to your development teams?

Understanding the way your development teams work and how they make decisions is critical. By understanding your team's general approach and development maturity, you can set realistic expectations of what you want them to do, determine how and where they can actually change, and see how you can best support and empower them. Without this, you will not be able to understand their point of view, creating friction and decreasing the chances of success. Also, remember that when thinking about the following topics, the answers will likely differ from team to team within the same business unit or even department. Be sure not to make assumptions about development teams and consider how they're likely to differ in their culture and approach.

What is their appetite for change?

Some development teams are attracted to the latest tech — sometimes too much — and are willing to try new tools, technologies, libraries, programming models, etc. just to stay modern or to see if it might be a better solution. Other teams only make these types of changes because of an existing pain point or because they are certain that the change will fix a problem that currently affects their application. And then there are other teams that will avoid change by default, operating under the philosophy of "if it's not broken, don't fix it."

Recognizing the behavior and mindset of your development teams in terms of how much they are willing to change will help to determine how best to collaborate with them in finding a common goal or destination for your AppSec program. For example, if they have resisted testing via Git pull requests (PRs) in the past, don't assume adding security testing in their PRs is going to be any different from previous attempts.

An important distinction to make when estimating the maturity and adaptability of your development teams is technical adoption vs. **process adoption**. It's common for less mature or earlier stage companies to be more focused on technical adoption in comparison to process standardization. For example, startups need to release fast to be first to market, which pushes process and standards maturation lower down on their priority list. It will be quite difficult to get a company in that phase to implement any security practice that acts as a gate.

How well defined is the team or projects?

A major difference between the security view and the developer view is ownership of assets. From the perspective of the security team, there are a number of assets owned by the engineering teams, some of which are more critical than others. From the developer perspective, each team has a focus, a number of projects or services that they own, projects they contribute to that other teams own, and projects they contribute to that many teams use and rely on, but don't have a specific team owning them.

With this in mind, asking a development team to own the security of their code and projects will have varying success depending on the ownership of each individual project. The more crisply defined the area is, the more likely a development team will be to take on the responsibility for the security of that project.

Additionally, the age of the team can also have an impact on the way you roll out. Ideally, standards and processes can be added from the start for a new team. But a more likely alternative is to take advantage of natural disruptions. For example, if a team has a major change unrelated to your program (new manager, team overhaul, etc.), they'll be more likely to adopt additional development standards as part of defining how they want to work.

What is your team's current capacity?

Like most teams, developers aren't waiting around idly for work to arrive. They're prioritizing what is most important for them to work on next, knowing full well that they can't get through everything on their ever-growing todo list. Simply adding more tasks to a developer or team's already overflowing plate is not constructive or supportive. This is amplified when teams are under-resourced and doing their best to go sprint after sprint while also trying to keep their heads above water.

> *We realize they have a lot of other responsibilities. They have to build features and products. They have to worry about performance reliability. We want to make participating in security as easy as possible."*
> — Jason Chan, VP Security at Netflix

What's the breadth of team abilities across the organization?

The reason to take the time to understand the dynamic of your development teams is because they're all different. It's extremely common to find a smaller number of teams on the leading edge that adopt newer technologies and processes first, with the rest of the teams in the organization following them over time. Majority adoption tends to happen slowly at first, but then ramps up when enough automations and best practices are in place to lower the barrier to adoption for other teams. Of course, requiring the bar to be significantly lower for broad adoption to occur means that many teams will take much longer to adopt — if at all. Before you can drive developer adoption, you'll need to know how many of your teams are going to need a lowered bar, as well as which teams can pilot to create the automations and practices.

An example of the breadth of team abilities can be seen in a team's tendencies to add integrations and to want to receive feedback. Typically, mature teams want feedback, and they want it as early as possible — in their IDE or in automations in their local build processes as well as in their Git repositories. Less mature teams may only want to automate in the CI process, meaning they'll get feedback late, typically just before a production deployment. Trying to add both of these types of teams in the same group of rollouts is unlikely to be successful, and most likely going to overwhelm the less mature teams.

A common cause of disparity across adoption is the complexity and age of the projects. An old, complex service that is being maintained rather than actively developed is a good example of a project that would not be a good candidate, as you can expect greater resistance when changing processes. Similarly, if your organization expanded inorganically (through acquisitions for example), you will be inheriting different technologies, pipelines, and cultures, which decreases the consistency across the wider organization, and increases the chances that team assumptions will be inaccurate.

How do your dev teams integrate security practices into their pipeline today?

Having taken time to assess the development teams in your organization, it's important to identify which security practices they currently follow and how they follow them, so you can determine a baseline to build from. Development teams typically start in a very hands-off way initially, perhaps periodically running tests that are for visibility purposes only, and integrating these into the pipeline where possible. There would likely be no blocking actions or gates anywhere to start with, allowing the development team to continue to release at the speed they're accustomed to.

Then take note of their preferences in integrations. For example, are they scanning or testing in CI, or earlier in PRs? Are they using an out-of-the-box integration, or have they done any scripting and automation to make it work for their pipeline or project? Do they test in IDEs or is their approach more reactive? A team that leans towards integration will be more likely to adopt an integrated developer security solution.

> *The key, from my perspective, is just understanding our engineering teams' preferred practices. What are those patterns so that we can partner to put in guardrails, rather than controls? We want to support the outcomes that the teams want. We need to be able to ensure that the appropriate practices and processes that they've determined are correct. And typically, we'll collaborate on those. If you really simplify it, the consistent thing is looking for gaps — gaps in our processes, gaps in our [collaboration]."*
>
> — Tim Crothers, SVP, Chief Security Officer at Mandiant

How do they prioritize security during sprints?

Another good indicator of where the team is on their security journey is whether their security focus is more on future development or their backlog. Often, backlogs can be quite overwhelming with thousands of issues or vulnerabilities, versus a new feature that might only introduce a few. An important aspect of this is also understanding how the team triages issues to understand what is important (or necessary) to fix, as well as when and how to escalate. If a team has OKRs around their backlog, as well as processes in place to create forward momentum, they'll be more likely to adopt tools that will help them get there faster.

How do they incorporate security testing and fixing into their sprints? Do they require a clean security test to deliver code, or do they add security tasks to their technical debt backlog and spend a sprint every few months fixing their issues? The latter is often common for teams that aren't quite as mature, or even some lower performing teams that can't contain the work in sprints. In these instances, it's equally common to see these types of dedicated sprints for scaling, performance, or reliability — all competing for time with security sprints.

Finally, does the team have any internal documentation of how they test, or SLAs they're working to for their issues? These are all good questions that don't just give you an understanding of where the team is today, but also tell you what the next step is to make it easier for the team to test and focus on the right things.

Are your developers empowered?

Developer empowerment is about whether or not a developer has the space to make their own decisions based on various inputs, to solve their own security issues, and ultimately take the responsibility and ownership of the security of their applications. **Empowerment requires support.** It's something that is given, grown, and supported by other teams. Let's start with the top-down support that's needed to empower developers to spend the time needed to deliver secure code.

> *If you read the Netflix culture memo on their website, one of the things that tends to pop out at people is this discussion of freedom and responsibility. And that is a concept that has driven how their engineering organization works. It is interesting because, as a human, we tend to index on the freedom side. We say, "Oh, this is great. I can do whatever I want. And I'm sure I'm going to have to be responsible for something. But we'll figure that out." In reality, there is a huge responsibility when you're building out software for a company like Netflix. You don't want that service to go down, so reliability matters deeply. You don't want it to be hacked, so security matters deeply. So somehow, you need to be very responsible as an engineer as well. And as a security organization, we often saw ourselves as there to help support and enable the business. What we wanted to do was allow the software engineers at the company to focus on the areas that they were hired for, effectively. But if they got to a place where security was really, really critical to what they were doing, then they would be leaning on us for help. And exactly where that boundary is becomes a judgment call, that if you have a good relationship between the security team and the engineers, you can figure that out."*
>
> — Bryan Payne, CISO at BetterUp, former Engineering Director,
> Product & Application Security at Netflix

What drives devs to spend time on security?

Whether developers want to spend time on security or not, they need to be given the time to do so, and in a way that allows it to be prioritized alongside other functional or non-functional deliverables. Developers should be able to decide how much time they need to invest in secure development practices, versus reliability or scaling considerations, based on the needs of the business. When they make these decisions, they should ultimately be able to state what they've done and achieved at their end of year review and be supported by their manager. It's up to managers to then reward them for their decisions, so developers know they've spent their time the right way. If this is not the case, the business is not empowering them to spend time on secure development.

Who should prioritize security for the team and business?

Ultimately, this comes from the very top: the CEO. If security is a business concern, it should be a CEO concern. This needs to trickle down the organization to the CIO and CTO, to the SVP/VPs of engineering, the directors, the managers and team leads, and ultimately to the developer actually running the tests and fixing issues. Once there is executive sponsorship, teams can dedicate a percentage of their sprints to engineering health, which includes security, rather than always being forced to focus on new features.

When this support and prioritization doesn't come from the top, it's extremely difficult to argue that security testing is as important (or possibly more important) than the next feature or a customer need. If these types of actions or activities are seen as extras that require justification, the default action becomes "do nothing", and discussions start with "why" rather than "how".

That's not to say that the CEO and developers are doing it for the same immediate reasons. A developer takes pride in their code. They want it to be accurate, fast, reliable, and secure. And the CEO cares about the brand and bottom line, and they worry about how damaging a security incident or data breach could be to their business. While both are good reasons to be secure, the fact remains that a top-down approach to security adoption will have more impact than just relying on developers to add more to their workload.

> *In terms of executive buy-in, [ours] came from the CISO, who had the CTO's ear. Consequently, it was really from the top-down in the technology org. Right from the top, there was the understanding and the need for security. In terms of implementation, you're dependent upon the software engineering organization and the leaders there."*
> — Nicholas Vinson, DevSecOps Lead at Pearson

Does your security team support your developers?

While it's the responsibility of the development team to secure their applications and code, they require support from the security team to do this well. When working collaboratively, the role of the security team moves away from an auditor position, where they run the tests and provide the results (and are seen as a blocker), to more of a security engineering team that helps the developers automate this into their processes, providing expert advice and consultation around the highest risk areas and prioritization.

The security team can't force developers to prioritize security work over their other tasks. As mentioned earlier, that's part of the overall engineering team and business priorities to decide. Historically, security teams have been looked at by developers as blockers, giving them security audits late in the development lifecycle. By empowering developers to become their own auditors, testing their code themselves, the security team can take a step back. They can then help development teams work through any results that require security expertise, shifting themselves from blocker to enabler. Additionally, this approach gives security the room they need to identify security champions with development teams, creating even more opportunities for collaboration.

Security can also educate the dev teams on vuln types and risk of exploits. This can be done in formal security champions programs, where other activities such as process roll out can also be well coordinated. While development teams work more independently, there still needs to be an overall governance structure and visibility for the business to understand its risks and exposures. The security team can provide the teams with guardrails backed with policies for their security testing, which can be applied to their pipelines and practices to help the development team with their early identification of issues and staying within their SLAs.

> *When I think about security champions and successful models, you actually identify several folks in engineering teams that are responsible for security. You build a scorecard that they are responsible for, that actually shows the things that we're doing to ensure that we're securing our product or feature in the right way. There's a path for engineers to talk with the central security organization as needed, and we're also providing them with the latest and greatest in training. If the security team is responsible for building tooling and things like that, those security champions are helping you go and drive adoption. So security champions have to be within the engineering organizations. They can't be sitting outside of it. They're really driving security."*
>
> — Rinki Sethi, VP and CISO at Bill.com

Another really important part of developer adoption is documentation. It's often written by developers, providing their side of the experience with usage advice, explanations of decisions, and nuances. The security team can help coordinate and share this documentation with other teams that are also onboarding similar practices, processes or tools, and also have a hand in their creation. Giving the development team a good self-serve experience, that they're familiar with all good developer tools, is a key aspect of successful adoption.

How are the security rules/process decisions in development workflows made?

One measure of empowerment is how much input and direction a developer or team has on their own processes and pipeline testing. When security rules and policies are made, it's understandable that they're driven with input from the security team, and then rolled out to development teams that take their advice on how to implement them into existing processes and roll out effective education. The important piece here is the way it is rolled out and implemented. The development teams need to be able to take ownership and make decisions and changes to their workflows the way they want. That's not to say every development team should do it in a custom way, as it's imperative teams learn best practices from each other. Development teams need the space to make decisions based on input from other development teams and the security group, such that the solution they put in place addresses the needs and standards that the security team requires.

One of the advantages of this approach is the dynamic it delivers. First of all, by not mandating a practice or process on the development team, you as a security individual can work with the development team to solve a security problem or requirement, rather than face the natural push back when something is imposed by an outsider. Having the security team take an advisory position offers a supporting function that is often more welcomed by development teams to help them make the right decisions.

Ultimately, this boils down to a question of ownership. If you try to have your security team own the security of the code being written by the development teams, you're asking them to scale their services very broadly across organizations and likely become a bottleneck. Traditionally, security tends to assume the role of a group that imposes requirements onto a development organization — often seen by developers as doing so without a mandate.

If you decide you do need to apply restrictions on technologies or a stack that you feel the development team should be using, it's important to create a paved road option that provides development teams with a choice of routes that are well supported by security and other teams. For example, it's a common practice to have a set of golden container images that have been fully tested by security and other teams, that developers can pick from and build upon. If you give options that also act as a paved road, you won't be seen as a blocker as you help teams stay secure.

Visibility and transparency across teams

Gaining visibility into the security posture of your applications, pipelines, and processes is the first step to identifying risk and exposures of your teams and business. However, not acting on the findings makes gaining visibility a pointless exercise. During this study, we found that using visibility to regularly summarize security posture in the form of an accountability scorecard or report was one of the defining factors of what made developer adoption successful.

The scorecards among those who achieved the best adoption covered security as well as many other aspects, including feature work, reliability, performance, and more. Their scorecards were generated monthly, showing operational metrics such as vulnerability counts, security tool adoption, project testing metrics, education levels, and much more. Very importantly, the specific metrics that each company included on their scorecards were aligned with business goals. Let's go into how these scorecards were used in more depth.

Prioritization and justification

With a scorecard that covers business metrics — including security — it's easy to see whether security is the most important thing to work on right now, or whether there are bigger problems elsewhere to address first. Scoring more than just security is important, as informed decisions can only be made when you have all the data at your disposal.

Scorecards should be generated at various levels, with the data tailored for the person consuming it. For example, an executive will want to see less detail than a team leader, as well as a closer connection to the overall business goals. The executive team can adjust their priorities based on the results of the report of where the business needs to apply greater focus. However, a team would want to know where they should spend more of their time, as well as how their scores compare to the rest of the organization.

All of this data allows developers and teams to better prioritize what they need to work on in upcoming sprints, as well as the justification for doing so. If anyone asks why a particular focus on security is happening, it's important to provide an objective scorecard rather than a subjective explanation.

As part of these scorecards, the security team should also provide specific prioritization recommendations for the development team. For example, they could highlight a list of the top vulnerabilities (or vulnerability types) to focus on to generate the greatest impact.

Accountability

For security to be a priority, there needs to be accountability from top to bottom. The CTO is accountable to the CEO, VPs of Engineering are accountable to CTOs, managers are accountable to VPs, and so forth all the way down to engineers. When everyone has a reason to care about security, it will be adopted as a standard part of any role.

Fortunately, scorecards are an easy and effective way to hold the right people accountable to the standards expected of them. By publishing scorecards, everyone within an organization can quickly know where they are in the red or are trending in the right direction. And when defined thresholds are met, those that are accountable for specific metrics can determine the causes, justifications, and possible solutions.

Just as we found with security prioritization, security accountability truly needs to go all the way to the top. If it doesn't, the message that it's important to the business (and therefore should be important to the development team) is heavily diluted and buy-in will be lost.

Lean on developer values and gamification

Typically, developers care about delivering an application that isn't just working, but is also fast, reliable, and secure. There are many things that can block a developer from delivering code that's up to their desired standard, but ultimately, they have pride in their deliverables. Scorecards are a great way to gamify this desire to deliver the best code possible, changing accountability from stick to carrot.

Making sure your development teams have visibility into the status of their projects is a good way to show them where they're going well or improving, as well as where they're struggling or falling behind. The pride a developer or development team has can take a hit if they see their scorecard is the worst in the department or significantly below the average for the business unit or overall company. They will naturally want to rectify this, but they need to be aware of it.

Gamification works well when it's done well. How you decide to do this will largely depend on your company culture, but sharing scorecards across teams and providing visibility prompts a gamification response. Nobody wants to be the worst team, and people enjoy seeing their scorecard progress and be above average for the local area — or even one of the best in the department. Equally important is the sharing of ideas and successes through announcements and discussions. These lead to new initiatives in teams wanting to replicate or push the ideas further in their areas.

Understanding the developer attitude to security

They do it because they should

The best approach for getting development teams to take on security is to make it an expected part of their role and goals. This gives development teams a mandate and a means to prioritize security. With this model, clear ownership is important so that teams aren't assuming another team has taken the responsibility for the security of a shared or legacy project.

Having people know what they should do isn't enough. It's just as important to have processes in place that help teams meet their security goals. For example, visibility into testing results, or an explanation of the additional levels of exposure and risk that exist after a developer has written a new feature. The visibility and transparency benefits of a scorecard can help provide the accountability that individual developers need to ensure it gets done as part of the workflow. This technique also works particularly well with outsourced teams that tend to float across projects and have less ownership over (or pride in) a project.

They do it because they should

It's such a good reason, we gave it twice! There's another driver behind why a developer should develop securely, and that's because it's the right thing to do. A proportion of your developers care about security or take great personal or professional pride and responsibility in coding in a sustainable and secure manner.

Make the right path the easy path (a.k.a. paved road)

Every obstacle that stands in the way of a developer performing a security task has the potential to stop them from completing that task. These obstacles can range from a lack of understanding to a painfully slow workflow to being overwhelmed with issues and not knowing where to start. Making security an easy add-on to existing workflows is key to the adoption of security practices.

Encouraging the use of standard practices via a paved road approach is a great way to provide helpful support and proven best practices. Additionally, there is a greater opportunity to create accurate and up-to-date documentation and advice that can be reused and updated by others regularly.

> *We call that concept a **paved road**. You could certainly bushwhack and make your way through the woods. But if you have this nice smooth paved road that gets you to your destination, you're likely to opt in there."*
> — Jason Chan, VP Security at Netflix

Think about what the developers need in order to make progress. Scanning, testing, and identifying issues is a relatively easy step in comparison to fixing. A developer doesn't want to know about the ten vulnerabilities in their dependency graph — they want to know that if they perform a minor upgrade to one dependency, they will fix their issues that are blocking their release. So think about solutions rather than problems, as the list is usually a lot smaller!

Also be very conscious of where developers want to spend their time, and how they want to consume security results and actions. Anything outside of their usual workflow is a distraction that they'll need to remember to do. Integrations into existing workflows, which have been implemented by people who understand the existing processes and technology underneath, can help create a great developer experience. Be intentional about making results useful, feedback clear and actionable, and keeping the workflow fast and painless. In short, implement with empathy for your developers.

Automate, automate, automate

The strongest DevOps pipelines are well-automated. In a recent study, Snyk found that automation also had a strong correlation with successful shift left security programs. The data showed that organizations with fully automated deployment pipelines are twice as likely to adopt static applications security testing (SAST) and software composition analysis (SCA) tooling into their SDLC. Additionally, those with full automation were over four times more likely to fix security issues in a day, and over twice as likely to fix them within a week.

It's clear that the more we automate, the more tests we can run, and the more issues we can catch. And with careful consideration, we can add security guardrails and policies on our pipelines to give us visibility and awareness of issues that need our attention and action. This provides a nice dynamic, as it's very familiar to a developer that already runs automations for things like integration testing.

Additionally, automating security also reduces the friction between the security team and the development team, because it's a process — not a team or an individual — saying you need to fix an issue. This allows the security team to support the development team and help with issues that the development team needs expertise with, rather than being a bottleneck or the bringer of bad news.

> *I would say the thing that they need to do is automate all of the security tools (that make sense) into their DevOps pipeline. That's something that I wish we had done earlier than we are doing it and have done it. Being able to, in an automated way, let a developer know as early as possible about a problem in their code is the thing that you have to do. If you could even do it at the code check-in point, so that they understand it then, or even with a tool as they're writing their code that flags something as a problem in their IDE immediately. Getting that automation in place is critical. Trying to fix an issue just before a product goes out the door is so much harder."*
> — Ryan Ware, Security Architect and director of the Intel Products Assurance and Security Tools team

Education and secure development knowledge

Education can be a hit-or-miss activity among development teams, particularly when there are so many irrelevant training courses and materials that teams are often mandated to take (and spend time taking a short exam on afterwards), which all likely ends up as a tick in a checkbox on an education compliance checklist.

Making education a relevant part of a developer's day, such that it helps them do their job, fix a bug, or develop more securely than they would have, increases the speed of security adoption. The impact of the course or education module needs to be of value to the developer, make them more secure, and be felt quickly.

When thinking about education, consider how it helps your teams focus on secure development. Think about the actions they will take away and actually apply to their processes or workflows. Think about how you can be targeted with your education to provide relevant assistance to them at the point in time when they need it most. Consider how your education goes beyond technology and also focuses on process and culture.

Here are some tips to get the most out of your security education:

- Training should be engaging enough that they want to do it. An easy way to boost the quality of your education is to run pilot programs, gather feedback, and make adjustments before rolling out to the masses.
- Recognize that development teams could have significant differences in their attack vectors, and provide relevant lessons accordingly. For example, directory traversal attacks may apply more to your backend developers.
- When training around vulnerabilities, ensure you train on those vulnerabilities that affect the team (based on applicable language, framework, libraries, etc.). Stories about incidents that affected other companies using similar ecosystems could help show the potential for disaster.
- While it's good to show real world examples of security issues, don't use fear as a main motivator. It might help you get movement in the short term, but it's not a long-term strategy. The exception being security issues that are truly scary — like ones that could devastate a business!
- Add education into scorecards to hold teams accountable to completing their education.
- Make sure your education focuses on how they can prevent tangible attacks. If you're explaining a vulnerability, show the vulnerability as exploitable. If you're explaining how to harden part of a pipeline, explain the attack surface that's being mitigated. Make it real and tangible.
- Ensure that documentation exists for any tasks that you want developers to carry out. Make sure developers validate the effectiveness of the docs, or better, write it themselves for the next developer or team. Remember, documentation isn't a replacement for guardrails, but you should also document your guardrails.
- If a red team finds an incident, use it as an opportunity to create relevant and realistic education. Providing deep dive information into the incident brings real risk to life. Bring the red team in for a Q&A session if possible.

Why do developers avoid security?

We've discussed why developers do take the time to build secure software. But it's equally important, if not more so, to understand why they push back and avoid taking additional steps to ensure they deliver secure code.

Friction

Depending on the stage of security adoption in your teams, and their maturity in modern development, the barriers that teams identify can vary. For teams that are early in their security onboarding, we see pushback when the reasons and value behind process changes aren't communicated clearly. The new process needs to be non-invasive and empathetic to the development team and their workflows. To drive adoption, over communicate, keep friction to a minimum by integrating with existing processes, and where possible, consolidate tools to reduce complexity.

> *I think one of the biggest challenges we've had is getting key stakeholders in the tech org and developers to really appreciate the importance and value of security. In an ideal world, there'd already be that understanding. A developer isn't really going to have much motivation to do something if they don't see the value in it. That's the same for a product owner or a dev manager or even a software engineering director. If they don't empathize with the value of security, they don't really have any impetus to prioritize [over] their feature work."*
>
> — Nicholas Vinson, DevSecOps Lead at Pearson

Mistrust

Providing value and maintaining or gaining trust is needed for a long-term, sustained process that will be followed by teams. When they lose trust in the data, tooling, or process, they will consider it a blocker that they will try to work around. For example, persistent false positives from a tool or process will not only dishearten or annoy the developer, but they will be more likely to ignore or write off the results of real vulnerabilities or issues than with data that they believe in.

Complexity

Similarly, when tools add complexity to existing problems and don't help to find or offer a solution, they just create more work for developers. Knowing a vulnerability exists is just the start of the problem. The developer still needs to identify all the ways and places that this vulnerability is accessible in the app (for example, the various dependency graph paths). Developers need to know instantly whether it can be fixed or not, and if it can, what the remediation process is. Developers want intuitive and instructive tooling so they can implement solutions quickly.

Ownership

Another reason why work might fall off one person's plate and not land on someone else's is ownership. With newer projects or features, the ownership might be clear, in that nobody else touches the code with the issue other than that one team. However, what if a part of the application was a common service that many teams used and contributed to, but nobody wholly owns? Who would pick up a bug or a security issue that doesn't affect them directly? Another common example would be a container image used by many different teams. When every team is scrambling to deliver their own team's feature work, shared common code is often a place where these issues aren't claimed.

Time

Of course, even where ownership is clear, there's still the challenge of finding time to include it into the workflow. Even if the automation is there to test your code, that's just the easy bit. The real work is in the developer consuming the results and performing the fix. The more that the tooling gives bad results or isn't prescriptive or autonomous with the remediation, the opportunity to drop it increases. Similarly, if the organization doesn't prioritize security fixes and activities, the developer will push back or ignore security teams asking them to test or fix.

Accountability

Accountability is a core pillar of why a developer should do security work, but who they are accountable to is equally important. For example, a developer would likely feel it more compelling to do work if their team or a security champion (developer) in their team or their management chain is holding them accountable to doing the work. Whereas. if it's just the security team holding them accountable, with no clear recourse, they'll feel less pressure to be secure.

Lack of role models

Humans model their behavior after those around them. If we don't see others developing securely, or notice other teams prioritizing security into their projects, it's easy to conform by ignoring security tasks, similarly. There are a couple of good ways to counteract this, including adding security visibility into code reviews so that security questions are being asked that require some thought or attention to the topic. Another way is to make role models!

Celebrate individuals for behavior that you want to see more of from the organization. Highlight where teams have excelled or made forward steps in integrating security into their pipeline or reducing their security backlog. Make sure the celebrations of security achievements are visible, done by the engineering leadership teams and repeatable by others through documentation, automation, etc.

How to improve security adoption in your development teams

So far we've heard a lot of accounts from various different groups and covered scenarios that both help and hinder developer adoption of security activities. In the remainder of this paper, we'll take a look at some actionable tips or mini-programs that you can consider applying in your organizations. Because these tips have come from a variety of companies at various stages of their digital and security transformations, they may not all be as relevant to you, so we'll start with a number of playbooks based on some of the most important considerations that would be applicable or most well-received by your development and security teams. Afterwards we'll go into more detail about some of the more tactical tips and approaches you can adopt or introduce in your teams.

Actionable playbooks

In this section we'll talk about trailblazing teams, adopter teams and inertia teams. Before we get started, let's define what we mean by each of these terms:

- **Trailblazing teams** tend to be your high performing teams who are the first to adopt processes and technologies because they are beneficial to the team or delivery. Typically these teams are a minority. Trailblazers are less likely to be overwhelmed, and more likely to discover and define best practices and standards for other teams in the organization to follow.
- **Adopter teams** tend to be solid delivery teams, adopting processes, techniques and technologies once they're proven by the trailblazing teams in the organization. These teams typically cover the majority of the development organization.
- **Inertia teams** resist change. They've done development a certain way for some time and are very comfortable with that approach and prefer to continue doing that because of familiarity and continuity.

In the table below, consider where your pain points are, whether cultural, process, or tooling based.

Actionable playbooks: Security champions (Culture)

Trailblazers	Adopters	Interias
When initially rolling out a security champions program, you should predominantly involve trailblazer teams. These are the teams you should use to help define the program with you. Use the trailblazers as a way to recruit security champions in the wider org. They should also have a big part in the content of what should be taught and where pain points exist.	The larger rollout of a security champions program will be among adopter teams. Using trailblazer champions to evangelize the program and benefits they are seeing is core to bringing adopters onboard. Listen to the adopter teams to understand how their problems differ from trailblazers, and try to pair them to solve problems together.	Don't expect people from the inertia teams to volunteer to join the group unless there are organizational pressures or rewards they consider valuable. Encourage adopter teams to champion security to these teams gently and show value and rewards they will be most interested in.

Actionable playbooks: AppSec tool (Tooling)

Trailblazers	Adopters	Interias
Onboarding of any new tool should start with trailblazing teams. They can help you identify the correct paths of interaction between the security team and development teams with the tool. They can also assess and advise where tools can best integrate into workflows and help with documentation, processes, and paved road style implementations that adopters can then use in a bigger scale rollout. Trailblazing teams are more capable of taking on new work and are less susceptible to being overwhelmed, so build integration options as far left as possible, including IDE, Git repo integration testing.	Once you have seen success from your rollouts to trailblazer teams, consider rolling out to a few adopters that are representative of the wider organization. Have them test out documentation and new workflows that were identified and created by trailblazers. Make changes as necessary and perform a wider rollout, keeping in mind that team capacity of additional work may be a greater challenge. Focus on future development testing first, rather than working on your backlog right away, to prevent overwhelming teams with new work. Also, be mindful of where adopter teams prefer to use tooling in their process and workflows and ensure their preferences and historical usage of integrations are taken into account.	During rollout, you will likely need to start further right, potentially at CI build time, if that's where the majority of their other tests are running. The earlier the testing, the more overwhelming the new results will be, which can increase frustration, pushback, or workarounds. Make sure documentation and paved path usage is very clear before trying to onboard the inertia teams, including support from the adopter champions and security teams as needed. Only focus on the most critical issues that tooling identifies initially and for new development features only, to reduce new work. Onboard much slower than you did with adopters and ramp up in the tool usage slowly, increasing guardrails over time.

Actionable playbooks: Celebrating success (Culture)

Trailblazers	Adopters	Interias
Trailblazers are leaders. When sharing their success, show them off as leading the way with new processes that improve the business. Show other teams how their adoption of new tools or techniques made their application development more secure. Highlighting how they are driving the engineering team forward is some of the most important recognition trailblazers can get, particularly from engineering leaders. Celebrating individuals is a useful way of driving other high performers. Swag always helps!	When celebrating adopters, it's less about new techniques, and more about how successful they've been with the adoption (and the fact that they have adopted). Focus more on implementation and adopter results. Make sure you celebrate team achievements, the integrations they've built up, and over time the number of issues they've fixed, etc. Use swag to get teams on board. Recognize teams at events or meetings in which many other teams are present, like monthly department or business unit meetings.	With inertia teams, lower the bar to what success looks like. Consider a no- or low-touch rollout as success and share with other inertia teams how easy it was. Be focused on the low-effort and high-value aspect of any new success. Work to build upon success in small steps to the next milestone. Learn what is valuable to your inertia teams as reward, as it can be different from adoption teams.

Actionable playbooks: Pipeline automation (Process/Tooling)

Trailblazers	Adopters	Interias
In order to build automated security testing into existing workflows, you need the trailblazer teams to be a big voice in where the automation should be and how they want to consume results. Depending on your team structure, it might be down to the trailblazer team to build the automation themselves, or work with a DevOps team to do so. With high performing teams, it's much more possible to push further left, and include security solutions into IDE, local build environments, and Git repositories. Make sure that enough is documented so that folks know what is running, the guardrails that are in place, and the processes for handling failures. Automation in the pipeline can also be used to push reporting and metrics, which can be led or helped by the trailblazing teams.	Adopter teams often take a subset of the automation points that trailblazers use based on how their existing stacks are built. For those who are using consistent stacks and pipelines, it's not uncommon to duplicate the automation points. Try not to mandate that teams have to add automation everywhere or even in most places, but make sure the value of adding the automation in various places is clear and that it's accepted by the development team. Initially, be sure that the automation has little or no impact on the developer workflow. Add it for visibility initially, and ramp up the policies, guardrails, and expectations of the development teams.	Depending on the inertia team you're dealing with, it's possible that you're more likely to get success further right in the pipeline, closer to the CI build process. Again, be sure to offer automations that can pull in security automation further left, but you need to match the team style with this. You want to be very careful about how you impact the developer workflow. So start off in a visibility-only mode, only moving (and doing so slowly) once the team gets more education around secure development processes.

Scorecards across teams should show similar data and exposures. Your expectations of what can be achieved by the various teams should differ. Use the data from the scorecards to show how activities from the teams translate to results, and then celebrate them.

Best practices to improve security

Let's dive into more detail around more tactical tips and best practices that you might want to consider when improving the security in your teams.

1. Security task forces

Create temporary task forces that are predominantly a group of developers, likely from cross-development teams, who all share a common goal of improving security adoption among developers. They help bridge the gap between developers and security by encouraging ownership of security in their teams through process and workflow changes. Note that this is a temporary group, and while it may share similar goals to a security champions program, it's not as formal nor as long-term in goals or strategy.

2.General security Slack/Teams channel

We all need another channel, right? Having a channel for security questions means people always have a place to go when they need help. This channel is often just triage, with questions being routed to the right person or place.

3.Top 3-5 vulnerabilities

Most projects will have a small number of vulnerabilities that are either more frequent, or more dangerous, than others. Rather than trying to train your development groups into being general security experts, provide quality training on the most common vulnerabilities that affect them. Be sure to share the risk and impact, which ultimately shows why this is so important to their specific project. If you can add examples, red team findings, or even production incidents, this brings weight to the importance of the issues.

4.Security champions program

If you are able to create a more formal program dedicated to improving the collaboration between development and security teams, this is an activity that has shown significant improvements in communication and rollout of security processes. For more information, read our deep dive on how others have seen success and failures in rolling out security champions programs to their organizations.

5.Make time for security

"Easier said than done," we hear you say! This isn't really a decision a developer should make. It's the top-down responsibility of the organization to empower the development teams to dedicate time to security, as well as other activities that contribute to general engineering health. Leadership can dedicate percentages of sprint time to these necessary activities, which engineering, product, and other teams respect the importance of.

> *There has to be a percent of job time dedicated to security. If it's just extracurricular work, it's never going to happen. That was actually one of the failures I had at [a previous company]. We didn't have buy-in from the top. I was working the program grounds up, and it just wasn't the right way to do it. As soon as we went to the top and said, "Okay. GM of this business or engineering head of this business, we want you to help us make this a part of the goals for the security champions," you saw a different accountability model, a different engagement back with the security organization to help craft what the model would even look like.*
>
> — Rinki Sethi, VP and CISO at Bill.com

6. Celebrate success

As mentioned in the previous section, celebrating success and creating security role models and best practices breeds further improvement among others. Whether during your security champions meetings, your security task force meetings, or even your department meetings, it's really important to shout loudly about the successes of your security rollouts. While it's important for the security team to help with this, it's most effective when the development leadership recognizes individuals and teams for their success and provides a platform to share new tools, processes or even interesting vulnerabilities they've identified or fixed.

> *We give out t-shirts that say, "Security Hero" on them. This is more exclusive, so it makes people want to step it up and really go above and beyond to make a security contribution. Maybe you eliminated cross-site scripting… you built it into the framework your team uses. Then that would warrant a "Security Hero" t-shirt. And then you get recognized in front of the whole company."*
> — Kyle Randolph, CISO at Verkada

> *Swag is key. And change it up, so you don't always give the same hoodie or the same sticker or the same t-shirt. Change it up, because if people expect, "If I do this I'm going to get this thing," it cheapens the experience. [Make it] somewhat of an unexpected surprise. They don't know when they're going to be rewarded, but they realize that there's a culture of recognition. When the software engineering team gets together, every couple weeks everybody gets together for an all hands. We will sit down with the security team and call out and publicly thank people for very specific actions. They're not asking us to do that, but it definitely goes a long way, and it promotes a cultural momentum that these are good things to do and that it is OK to take the time to produce better quality code. I'm an evangelist about that."*
> — Zach Powers, CISO at Benchling

7. Communicate, communicate, and communicate

We're saying it three times because it's only when you think you're overcommunicating that you are communicating at the right level. Make sure you're keeping the development teams in the know when new changes roll out, why they're happening, and what it's going to look like, before it happens. This could involve internal blog posts describing the new changes, demos that help with internal enablement, and of course, lots of reference documentation to help support your teams. If there are any problems, be sure to fail fast, fail forward, fix the issue, and continue with the rollout. And while you're communicating, be sure to also listen.

8. Build trust, not fear

In order to get someone else to do something, it's tempting to share the impact of what would happen if that thing wasn't done. While it is important to understand risk and use that as part of a prioritization exercise, it's not a useful stick to use over and over again. To build up trust, work with the development team to build up and jointly understand the risk; be proactive about measures, whether tooling or advice; and start slowly in a diplomatic way. Trust takes time and a shared goal which you both agree on and work towards to make sure that you're not doing things because they're mandatory, but rather doing things because they're necessary. The difference is subtle but extremely important. The security team is there to make the development process and work easier, not be a blocker or an annoyance.

Techniques to improve adoption

In addition to some of the bigger tips and programs mentioned above, there are additional techniques that can be used to improve the developer experience while performing security tasks. Here are some of the tips shared by those achieving strong developer adoption:

1. Make the right thing the easy thing

There are many good reasons why developers should spend time on security issues or tasks. However, all it takes is for a process to be complex, or even just a little unclear, for someone to avoid doing it or put it off for later. Making the path easy, predictable, and achievable — in as much of a self-serve way as possible — can really help developers do the right thing, easily. Examples might include providing a template in Jira that developers can clone that has clearly defined issues or tasks that need to be completed.

> *We developed our automation, deployed it, and then I went on paternity leave for a while. And then COVID happened. We didn't really advertise the automation or push people to use it, but what we found was there was roughly 60% rate across teams, where around 65% of the PRs that we created so far have been merged and closed, which is a significant number considering we hadn't sent any communications out to developers regarding this. This tells you that everyone wants to do the right thing, but maybe doesn't have time to do it. So when you make it as simple as possible, they do the right thing."*
>
> — Yashvier Kosaraju, VP of Security, Compliance & IT at Sendbird

2. Good Documentation

Never leave your developers stranded in a place where they don't know what to do, don't understand the process, or don't know how they should use their tools. Good documentation is paramount to successful onboarding and a smooth adoption that is more self-serve, rather than needing individual from another person or team. It's important that when teams use documentation, they keep it up to date. As teams build new processes or automations, it's important that they document it for other teams to follow. Asking developers to help with the documentation to make sure it meets their needs and is accurate is essential in making it relevant.

3. Automate, automate, automate!

To make it easier for developers to adopt new tools and practices, make it an automated part of the developer workflow. Additionally, automate it in the right place for the team. If your team likes GitHub actions, make sure the tool runs as a GitHub action. If they love feedback in IDEs, make sure they're aware of the tools IDE plugins.

4. Use tools that are designed for developers

If your development team is really struggling with running tests, understanding results, or fixing their security issues, it could well be because the tooling they're using is not designed for them. Traditionally, the security team would perform security testing and audit a codebase or application, giving the development team the results in a problem-oriented way. This creates friction for a developer who is much more familiar with a developer tool that integrates with their existing tool set, that they are most familiar with, which produces results that have suggested solutions they can take to remediate the issues. This is what empowers developers to fix. There's more on what makes developer tools in the next section.

5. Consolidate tools

If your developers can use one tool or vendor for multiple types of security risks, it helps with consistency and adoption. For example, if a developer is in their IDE and has some Java files open, a pom.xml build file, a Dockerfile, and a Terraform script, do you think they want them to run four tools to test for vulnerabilities and issues, or just one?

6. Identify ownership

Ownership is always important, although this time we're not talking about whether security or development own a task or responsibility. It's more about which development team should own an issue. It's easier for developers who work on a project or piece of code than someone not familiar with it. Try to tag projects with owners so that issues can be routed to the right people in the development teams. This is most important for projects that are shared and used by multiple teams.

7. Support your developers

The days when security teams could pass issues to a developer and expect them to fix them by themselves are behind us. Security teams need to be there to help when security expertise is needed, and to aid developers with issue resolution where required. Working as one team and sharing not just goals but also problems and solutions is key to the success of a developer-first security program and its adoption.

8. Avoid overwhelming developers

Depending on the maturity of your development teams and developers, each will want to consume security activities at different levels to avoid being overwhelmed by issues or activities. For example, a team which is only just starting on their security journey would be overwhelmed seeing 100,000 vulnerabilities in their backlog. It's better if they start by only looking at the security of the changes they're making, to start off with, and likely just the highest most critical issues that are being introduced in those changes. The threshold should be movable based on your team's capabilities and capacity. For those that are more advanced and ready to look at backlogs and improve the overall security of their projects, it's important to prioritize issues so that they see the top five or ten issues they should fix, rather than the 100,000 that they don't want to know about right away. This should be an exercise of prioritization and triage that the security team and development team leads or security champions can work on together.

What makes a tool a developer tool?

The way to scale security in organizations is to empower developers to take responsibility for the security of the applications they are creating. This shift in responsibility and attitude to developer security in our culture needs to be matched in our processes and of course our tooling. Tools need to shift to be developer-friendly and provide what developers need for them to do their job, develop and deliver code securely.

The big difference between what developers and security teams want from a tool is intent. Traditional security teams are auditors. They want to test, report, and provide developers with a list of problems. Developers don't want this. Developers want a tool that they are familiar with, that acts predictably, and gives them actionable solutions instead of more problems to research.

Here are a few of the common traits of a great developer tool:

- **Successful self-serve adoption**
  Practically all successful developer tools have great self-serve usage, including easy onboarding, intuitive workflows, and great documentation. This is the way developers like to consume tools, and it forces vendors to ensure that their tools relate to developers without a salesperson pushing it. Always look for tools with a proven track record of self-serve adoption by developers.
- **Seamless integration into workflows**
  Developer tools, for the most part, look to meet developers where they are instead of asking them to open yet another tool. They integrate elegantly into their IDE, Git, and pipelines and provide value at the right point in time. Integrations are about more than technical hooks — they need to adapt to workflows and best practices or they will be rejected.
- **Rich APIs and automation-friendly**
  Although opinionated integrations are needed to get started, a developer tool must also be a Swiss Army knife. A rich API and automation-friendly client (CLI/SDK) are mandatory, both to tune the tool to every pipeline and to allow the community to build on it. If you can't build on a tool, it's not a great developer tool.
- **Adoption by open source and community**
  Developers look to fellow developers to validate a tool's credibility, and open source adoption is the best indicator of that. Seeing open source projects integrate a security tool or finding open source projects that are built on it are great developer community validations. When inspecting a security tool, inspect its adoption amid open source and draw your own conclusions.
- **Expert guidance, automatic fixes**
  A great developer tool makes it easy for a developer to do the right thing, even when they don't know what the right thing is. For example, any linter will find bad code, but a good linter will tell a developer how to fix it. A greater linter will apply fixes with a click or even automatically.

> *The traditional way [to handle application security] has been to create tickets that go into the backlog and then have your management team, which owns an SLA, talk to the team. But there is friction in that. The easier you can make it for someone to be secure, the more likely they do it. When we think about the tools that we want to provide to developers, the first questions we ask ourselves are, "how can we make this easy for them to use?" and "how can we make it easy for them to fix the findings?" Tickets are nice for tracking, but they don't make it easier for [developers]. What makes it easy for them are comments and PRs. Or a PR to fix the issue itself. Or [guidance] that tells them what they need to do. That is the philosophy we have been using"*
> — Yashvier Kosaraju, VP of Security, Compliance & IT at Sendbird.

- **Application context**
  Developers have a never-ending checklist, so a developer tool needs to help them prioritize their work. This means providing the context to know what's critical and what can wait, as well as how work fits into the bigger application picture. If the action items from different tools should be prioritized against one another, sharing a backlog can increase efficiency and results

Conclusion: Developer adoption is key to security

To summarize, developer adoption is a core goal that helps us as a business to improve our overall security posture. To achieve strong and natural adoption, it's important to create a collaborative culture in which your security team and development teams talk the same language, working together to achieve shared goals. The security team is no longer there to audit and give more work to the engineering teams. They're there to support and enable engineers to find and tackle security issues as early, quickly and effectively as possible. Engineering teams need to see security teams and the group that empowers them to achieve that, and they should reach out for help when that isn't the case.

This paper covered three areas of change providing example activities that we have seen work effectively to improve developer adoption: Culture, Process, and Tooling. With all three of these, it's essential to involve both development and security teams in the creation or changes to existing processes, programs or tooling choices. Furthermore, take time to learn about your development organization, how they like to work, by team, and build a strategy on how to deal with the variance of team maturities and performance.

As you go forward in adopting or trying out ideas from this paper in your organization, be sure not just to involve your wider teams as part of the planning and decision making process, but be very intentional about the speed with which you roll out the new initiative. Make sure at all times that you're not overwhelming your engineers, but rather getting them to adopt at the speed with which they have capacity for, so that they can build a secure development muscle sustainably. Good luck!