CHAPTER 21

Building a Culture of Security and Reliability

By Heather Adkins
with Peter Valchev, Felix Gröbert, Ana Oprea,
Sergey Simakov, Douglas Colish, and Betsy Beyer

Imagine a new employee joining your organization. Is that employee aware of the importance of security and reliability, and how they are prioritized among other organizational objectives? Do employees know what role they play in ensuring that systems can withstand errors coming from a bad push or malicious adversaries?

This chapter outlines aspects of a healthy culture of security and reliability. We also touch on how you can affect organizational culture by choosing good practices when it's time to make changes. Finally, we provide insights into how to influence leadership and teams across the organization to build buy-in for security and reliability.

We share some practices here that we've found useful at Google and elsewhere, but keep in mind that no two organizations are exactly the same—you'll need to adapt these techniques to the culture of your organization. You will also find that it's likely not possible to adopt all of these strategies. This chapter is meant to be a guide and reference for continuous consideration. It's worth noting that at Google, we don't practice the recommendations we cover here perfectly every day, but rather seek to improve in each approach as part of building our overall culture.

Effective security and reliability flourish when organizations embrace their importance by building a culture around these fundamentals. Organizations that explicitly design, implement, and maintain the culture they seek to embody achieve success by making culture a team effort—the responsibility of everyone, from the CEO and their

leadership team, to technical leaders and managers, to the people who design, implement, and maintain systems.

Imagine this scenario: just last week, the CEO told your entire organization that getting the next Big Deal was critical to the future of the company. This afternoon, you found evidence of an attacker on the company's systems, and you know these systems will have to be taken offline. Customers are going to be angry, and the Big Deal may be at risk. You also know that your team may get blamed for not applying security patches last month, but a lot of people were on vacation and everyone is under tight deadlines for the Big Deal. What kinds of decisions does your company culture support employees making in this situation? A healthy organization with a strong security culture would encourage employees to report the incidents immediately despite the risk of delaying the Big Deal.

Suppose that at the same time you're investigating the malicious interloper, the frontend development team accidently pushes a significant change intended for staging to the live production system. The error takes the company's revenue stream offline for over an hour, and customers are overwhelming the support helpline. The trust of your customer base is quickly eroding. A culture of reliability would encourage employees to redesign the process that allowed an accidental frontend push, so that teams can manage the needs of customers and the risk of missing or delaying the Big Deal.

In these situations, cultural norms should encourage blameless postmortems to uncover patterns of failure that can be fixed, thereby avoiding harmful conditions in the future.[1] Companies with healthy cultures know that getting hacked once is very painful, but getting hacked twice is even worse. Similarly, they know that 100% is never the right reliability target; using tools like error budgets[2] and controls around safe code pushes can keep users happy by striking the right balance between reliability and velocity. Finally, companies with a healthy security and reliability culture know that in the long term, customers appreciate transparency when incidents inevitably occur, and that hiding such incidents can erode user trust.

This chapter describes some patterns and anti-patterns for building a culture of security and reliability. While we hope that this information will be helpful to organizations of all shapes and sizes, culture is a unique element of an organization, crafted in the context of its particular challenges and features. No two organizations will have the same culture, and not all of the advice we provide here may be applicable to everyone. This is a chapter meant to provide a range of ideas on the topic of culture, but it's unlikely that every organization will be able to adopt all of the practices we

1 See Chapter 15 of the SRE book.

2 This is discussed in Chapter 3 of the SRE book.

discuss. The somewhat idealized view we present here won't be wholly practical in real-world situations. At Google too, we don't get culture right all the time, and we're constantly seeking to improve upon the status quo. We hope that from among the wide range of viewpoints and options presented here, you'll find some that may work in your environment.

Defining a Healthy Security and Reliability Culture

Like healthy systems, a healthy team culture can be explicitly designed, implemented, and maintained. Throughout this book, we've focused on the technical and process components of building healthy systems. Design principles for building healthy cultures exist as well. In fact, culture is a core component of designing, implementing, and maintaining secure and reliable systems.

Culture of Security and Reliability by Default

As we discuss in Chapter 4, it's often tempting to delay considering security and reliability until the later stages of a project's lifecycle. This postponement appears to accelerate initial velocity, but it does so at the expense of sustained velocity and a potentially increased cost of retrofits. Over time, these retrofits can increase technical debt or be applied inconsistently, leading to failure. To illustrate this point, imagine that when buying a car, you had to separately seek out a seat belt vendor, someone to review the safety of the car's windshield, and an inspector to validate the airbags. Addressing safety and reliability concerns only *after* manufacturing a car would place a high burden on the consumer, who may not be in the best position to assess whether the solutions implemented are sufficient. It could also lead to inconsistent practices between any two manufactured cars.

This analogy reflects the need for systems to be *secure and reliable by default*. When security and reliability choices are made throughout the lifecycle of a project, it's easier to maintain consistency. Also, when these are integrated parts of the system, they can become invisible to the consumer. To return to the car analogy: consumers don't need to give much thought to safety mechanisms like seat belts, windshields, or rearview cameras to trust that they do the right thing.

Organizations with healthy cultures of by-default security and reliability encourage employees to discuss such topics early in the project lifecycle—for example, during the design stage—and throughout each iteration of implementation. As products mature, their security and reliability will continue to mature organically as well; this is formalized into the software development lifecycle.

Such a culture makes it easier for people designing, maintaining, and implementing systems to incorporate the themes of security and reliability automatically and transparently. For example, you can introduce automation for continuous builds,

sanitizers, vulnerability discovery, and testing. Application frameworks and common libraries can help developers avoid common vulnerabilities like XSS and SQL injection. Guidance around choosing the appropriate programming languages or programming language features can help avoid memory corruption errors. This kind of automatic security aims to reduce friction (such as slow code audits) and errors (bugs not spotted during review), and should be relatively transparent to developers. As systems mature within these security and reliability constructs, ideally, employees will increasingly trust these implementations.

We provide some insight into the evolution of creating a culture of security and reliability by default at Google in Chapters 12, 13, and 19.

Culture of Review

When a strong review culture is in place, everyone is encouraged to think ahead of time about their role in approving changes. This can bolster the ongoing security and reliability properties of the system by ensuring that changes take these special considerations into account. Peer reviews to ensure the security and reliability features of a system apply in a variety of change scenarios, such as these:

- Multi-party authorization reviews for access or changes to maintain least privilege (see Chapter 5)
- Peer reviews to ensure that code changes are appropriate and of high quality, including security and reliability considerations (see Chapter 12)
- Peer reviews of configuration changes before they are pushed to production systems (see Chapter 14)

Building such a culture requires a broad understanding across the organization about the value of the reviews and how to carry them out.

Change review practices should be documented to set clear expectations about what will happen during the review. For example, for peer code reviews, you might document the organization's engineering practices related to code review and educate all new developers about these expectations.[3] When requiring peer review for multi-party authorization schemes (as described in Chapter 5), document when access will be granted and under what conditions it might be refused. This establishes a common cultural set of expectations across the organization, so that only valid approval requests will succeed. Similarly, you should set expectations that if an approver

3 Google's practices for code reviews are documented in the Code Review Developer Guide. For additional background on Google's code review process and culture, see Sadowski, Caitlin et al. 2018. "Modern Code Review: A Case Study at Google." *https://oreil.ly/IfFJ1*.

denies a request, the reasons are understandable based on the documented policy, to avoid hard feelings between people and creating an "us versus them" mentality.

The corollary to documentation is to educate reviewers so they understand the baseline expectations for being a reviewer. This education can happen early on, during onboarding to the organization or a project. Consider onboarding new reviewers through an apprenticeship scheme, in which a more experienced or calibrated reviewer also looks over the change.

A culture of review requires everyone to participate in the review processes. While owners are responsible for ensuring the overall direction and standards of their respective areas, they too should be individually held accountable for the changes they initiate. No one should be able to opt out of a review simply because they have a senior role or don't want to participate. The owner of a code tree isn't exempt from having their code and configuration changes reviewed. In the same way, system owners aren't exempted from participating in multi-party authorization for logins.

Reducing Friction

Consider establishing lightweight review options that enable a quicker change process for minor changes. As we state in Chapter 14, reviews are effective only if they're mandatory. However, reviews need not be burdensome. A minor configuration change or a multi-party authorization request to temporarily access a nonsensitive resource doesn't have to require a heavyweight process. Be sure to have clearly established guidelines about what needs a lightweight review and what needs a heavyweight one, so developers and reviewers have the same expectations.

By choosing the right workflow tooling, you can also help reduce friction for developers—tools that make it easy to initiate changes, provide diffs for reviewers, and have automation support for small changes can decrease friction.[4]

Ensure that reviewers have the context necessary to make decisions, and have the option to decline or redirect a review if they lack enough context to accurately assess whether the change is safe. This is especially important when reviewing the security and reliability properties of an access request, such as multi-party authorization, or changes to code snippets with safety implications. If the reviewer is not familiar with the kinds of pitfalls to look out for, then the review will not act as a sufficient control. Automated checks can assist with building this context. For example, in Chapter 13 we discussed how Google uses Tricorder to automatically raise security issues for developers and reviewers in the presubmit phase of code changes.

4 A 2018 study of the modern code review process at Google found that developers valued the low-friction workflows provided by their tooling.

Culture of Awareness

When members of an organization are aware they have security and reliability responsibilities, and know how to carry them out, they can be effective in achieving good outcomes. For example, an engineer may need to take extra steps to keep their account secure because they access sensitive systems. Someone whose job entails frequent communication with external parties may receive more phishing emails. An executive may be at higher risk when they travel to certain parts of the world. Organizations with healthy cultures build awareness of these kinds of conditions and reinforce it through educational programs.

Awareness and education strategies are key to building a strong security culture. These initiatives should strive to be lighthearted and fun so that learners are interested in the content. People retain different types of information at different rates depending on how that information is conveyed, their existing familiarity with the material, and even personal factors like age and background. In our experience, many learners have a higher retention rate with interactive methods of learning like hands-on labs than with passive learning methods like watching a video. When building awareness, optimize for the best learning experience by carefully considering what types of information you want people to retain and how you want them to learn.

Google takes several approaches to educating employees about security and reliability. On a broad scale, we provide mandatory annual training for all employees. We then reinforce these messages through specialty programs for certain roles. Here are some tactics we've found useful over years of implementing these programs at Google:

Interactive talks

Talks that encourage audience participation can be a way of relaying complex information in a compelling way. For example, at Google, sharing top root causes and mitigations for significant security and reliability incidents has helped our employees better understand why we focus on these topics. We've found that these types of interactive discussions also encourage people to raise issues they find, from suspicious activity on their workstations to buggy code that could take systems down. This practice helps people feel like they're part of the team that makes the organization more reliable and secure.

Games

Gamifying security and reliability is another way to build awareness. These methods tend to scale more effectively to larger organizations, which may be better able to give players flexibility around when they take the training and an option to retake it if they want. Our XSS game (shown in Figure 21-1) has been quite successful in teaching developers about this common web application vulnerability.

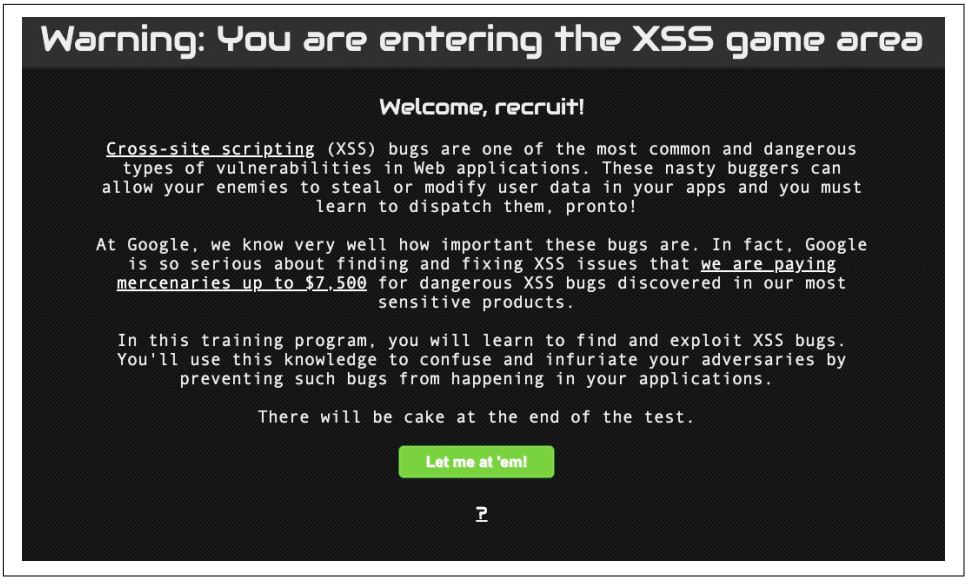

Figure 21-1. A security training game

Reference documentation
: Reading documentation may have a lower retention rate for learning than methods like hands-on exercises, but we've found that it's critically important to provide developers with strong documentation they can reference when needed. As we note in Chapter 12, reference documentation is important because it's difficult to keep the numerous nuances of security and reliability in one's mind simultaneously. For guidance on common security problems, Google maintains a set of internal security best practices that engineers can search for answers to problems as they arise.[5] All documentation should have clear ownership and be kept up to date or deprecated when it's not relevant anymore.

Awareness campaigns
: It can be hard to notify developers about recent security and reliability issues and developments. To tackle this, Google publishes weekly engineering advice in a one-page format (Figure 21-2 shows an example). These "Testing on the Toilet" episodes are distributed to restrooms throughout all Google offices. While initially aimed at improving testing, the program also occasionally features security

5 For example, Google maintains a set of best practices for cross-site scripting.

and reliability topics. A flyer posted in an unavoidable location is a good way to present tips and provide inspiration.[6]

Figure 21-2. An episode of Testing on the Toilet

Just-in-time notifications

Especially during procedural steps (like checking in code or upgrading software), you may want to remind people of good security and reliability practices. Showing just-in-time notifications to people in these situations can help them make better risk decisions. At Google, we have experimented with showing pop-ups and hints in the moment—for example, when employees are upgrading software from an untrusted repository or trying to upload sensitive data to unapproved cloud storage systems. When users see alerts in real time when it matters, they

6 A recent study on the use of the Testing-on-the-Toilet program at Google showed that the program increased developer awareness. See Murphy-Hill, Emerson et al. 2019. "Do Developers Learn New Tools on the Toilet?" *Proceedings of the 41st International Conference on Software Engineering*. https://oreil.ly/ZN18B.

can make better decisions for themselves and avoid mistakes.[7] On a related note, as we discuss in Chapter 13, presenting developers with presubmit security and reliability hints helps them make better choices when developing code.

Culture of Yes

Over time, organizations can develop a conservative risk culture, especially if security breaches or reliability issues have led to revenue loss or other bad outcomes. In an extreme form, this mindset can lead to a *culture of no*: the inclination to avoid risky changes and the negative consequences they might entail. When perpetuated in the name of security or reliability, a culture of no can cause an organization to stagnate, and even fail to innovate. We've found that healthy organizations have a way to work through the challenge of saying "yes" when taking advantage of an opportunity requires some amount of risk—that is, to take risks deliberately. To embrace risk in this way, you typically need to be able to assess and measure it.

As a concrete example, in Chapter 8 we describe the approach used to secure Google App Engine, a platform that proposed running third-party unverified code. In this situation, Google's security team could have judged the launch as too risky. After all, running arbitrary untrusted code is a fairly well-known security risk. You don't know, for instance, if the third party managing the code might be malicious and try to escape from the platform's execution environment and compromise your infrastructure. To address this risk, we embarked on an ambitious collaboration between the product and security teams to produce a layered, hardened system, which allowed us to launch a product that would have otherwise seemed too dangerous. This kind of collaboration made it easier to build additional security into the platform over time, since there was an established base of trust between the teams.

Balancing Accountability and Risk Taking

The inclination to say no to risky changes can also stem from an individual's fear of making the wrong call. In the context of design reviews and code audits, this tendency can be especially challenging, since it's not possible for humans performing manual reviews to find every single security or reliability problem. If people think they are the only line of defense against a risky change, they may be less inclined to embrace risk.

7 This topic is closely related to *nudging*, a method of changing behavior by subtly encouraging people to do the right thing. Nudge theory was developed by Richard Thaler and Cass Sunstein, who were awarded a Nobel Prize in Economics for their contribution to behavioral economics. For more information, see Thaler, Richard H., and Cass R. Sunstein. 2008. *Nudge: Improving Decisions About Health, Wealth, and Happiness*. New Haven, CT: Yale University Press.

> Organizations that embrace risk defend against this pressure by adopting design strategies for elements such as least privilege, resiliency, and testing. These layered approaches lift the burden of being perfect off of individuals' shoulders. If someone makes a mistake during review, other checks and balances can prevent disaster.

Another approach to embracing risk is to use error budgets, which allow for failures up to a certain limit. Once the organization reaches a predetermined maximum limit, teams are expected to collaborate to reduce risk to normal levels. Because the error budget maintains an emphasis on security and reliability throughout the product's lifecycle, innovators have the freedom to introduce a certain number of risky changes.

Culture of Inevitably

No system is perfect, and any system may eventually fail. At some point, your organization will likely experience a service outage or a security incident. Embracing this inevitability can help teams have the appropriate frame of mind to build secure and reliable systems and respond to failures.[8] At Google, we assume failure can happen at any time—not because we aren't diligent about proactive measures or because we lack confidence in our systems, but because we know that real-world systems can never be 100% secure and reliable.

Chapter 16 discusses the need to prepare for the inevitable. Teams that embrace a culture of inevitability dedicate time to prepare for disasters so they can respond effectively. They talk openly about the possibilities for failure and set time aside to simulate these scenarios. Incident response skills are effective only when you use them regularly, so it's a good idea to use exercises such as tabletops, Red Team attacks, hands-on recovery tests, and disaster role playing to test and refine your organization's processes. Organizations that embrace the inevitable also study any failures that do occur, including within their peer groups. Internally, they use blameless postmortems—discussed in Chapter 18 of this book and in Chapter 15 of the SRE book—to reduce the fear of failure and build confidence that repeated events will be unlikely. They also make use of after-action reports published by other organizations, both within and outside of their industry. These reports provide a broader understanding of failure scenarios that may be relevant to the organization.

8 Dave Rensin, Director of Customer Reliability Engineering at Google, considers this topic in greater detail in his talk "Less Risk Through Greater Humanity".

Using Related Case Studies

In 2003, NASA convened the Columbia Disaster Investigation Board, which aimed to understand the root causes of the Space Shuttle Columbia incident.[9] The final report is a useful example of how an organization can study the underpinnings of failure to effect significant changes that contribute to a culture of safety and reliability. Specifically, Chapter 7 of the report notes that NASA's organizational setup and norms had impeded a culture of safety.

This publicly available real-world example demonstrates good use of after-action reports, which we recommend conducting as blameless postmortems. These kinds of reports can also give leadership a clear understanding of how cultural breakdowns can lead to reliability and security issues.

Culture of Sustainability

In order to sustain the reliability and security features of a system in the long term, your organization must ensure that efforts to improve them are made continuously—and dedicate sufficient resources (staff and time) to the task. Sustainability requires building the means to handle outages, security incidents, and other emergencies on an ongoing basis, using clearly defined processes.

To maintain this effort, teams must be able to balance the time spent on reactive work versus proactive investments that will pay off over the long term. To recall our example of the California Department of Forestry and Fire Protection from Chapter 17, effective teams allocate the burden of hard work across many people's shoulders so that no one person is saddled with excessive responsibility.

Organizations with a *culture of sustainability* measure the workloads necessary to handle operational work (for example, incident response[10]) as well as the investments required to make improvements over time.[11] They consider stress, burnout, and morale in their planning, adding sufficient resources to sustain long-term efforts or deferring work where necessary through prioritization. They avoid the need for heroics by setting up repeatable and predictable processes for handling emergencies and rotating the staff who handle emergencies regularly. They also proactively handle morale issues by surfacing individuals' concerns and continuously motivating people.

9 The final report of the Columbia Disaster Investigation Board is preserved on the NASA website for the general public to read. Chapter 7 in particular focuses on the culture of safety at NASA and its impact on the disaster. We've found that the report's findings can often be extrapolated to organizational culture in other types of engineering organizations.

10 See Chapter 29 of the SRE book.

11 See Chapter 32 of the SRE book.

Sustainable Reliability and Security Culture at Google

Google has many moving parts and a correspondingly complex adversary landscape, which necessitates large teams of dedicated SREs and operational security personnel. These teams operate in "follow the sun" shifts in order to avoid stress and burnout, and are designed to be staffed accordingly. They spend only a portion of their time doing operational work and the rest making infrastructure improvements. We also designate dedicated development resources for long-term initiatives. There's an intended virtuous circle in this work because these initiatives strive to automate away toilsome operational work, improving morale and freeing people to spend more time on mentally challenging problems.

The linchpin of the sustainability effort at Google, however, is the way we manage the health of our people. Managers are trained to focus on building and maintaining healthy teams, which includes addressing negative impacts of the work. These principles are encoded in our culture as part of Project Oxygen.

Having a culture of sustainably also means knowing that sometimes exceptional circumstances can cause temporary deviations from expected workloads, and having good processes to handle those deviations. For example, if multiple business-critical systems haven't met their SLOs for a long period of time, or a serious security breach results in an extraordinary response effort, you may need an "all hands on deck" effort to get things back on track. During this time, teams may be entirely devoted to operational work and improving security or reliability. While you might have to defer all other organizational work, you may also have to deviate from best practices.

In healthy organizations, such extraordinary disruptions of normal business operations should be rare. When navigating these situations, the following considerations can help maintain a culture of sustainability once the situation is resolved:

- When operating outside of normal operations, be sure to clarify that the situation is temporary. For example, if you require all developers to manually supervise the changes they push to production (instead of only using automated systems), this might cause a lot of toil and unhappiness in the long term. Make clear that you expect the situation to return to normal soon, and give an idea of when that will happen.
- Have a dedicated group on standby that has an understanding of the risk landscape and the authority to make decisions quickly. For example, this group might grant exceptions to standard security and reliability procedures. This will reduce friction in execution, while giving the organization some assurance that safety mechanisms are still in place. Have a way to flag the times you had to bypass or overturn best practices, and be sure to address those one-offs later.

- When the event is complete, be sure that your postmortem reviews the reward system that may have led to the emergency. Sometimes, cultural issues like prioritizing feature launch over reliability or security features can lead to a build-up of technical debt. Addressing such issues proactively going forward will help the organization return to a cadence of sustainability.

Changing Culture Through Good Practice

Affecting organizational culture can be difficult, especially if the team or project you're working on is well established. It's not uncommon for an organization to want to make security and reliability improvements, but find that cultural obstacles stand in the way. Counterproductive cultures exist for many reasons: leadership approaches, starvation of resources, and more.

A common element of resistance to change—the kind of change necessary for security and reliability improvements—is fear. Change can conjure images of chaos, greater friction, loss of productivity and control, and increased risk. In particular, the topic of friction frequently surfaces in relation to new reliability and security controls. New access checks, processes, and procedures can be interpreted as interfering with developer or operational productivity. When organizations face tight deadlines and high expectations to deliver, either self-imposed or driven by management, fear of these new controls can heighten concerns. However, in our opinion, the belief that security and reliability improvements have to create friction is a myth. If you implement change with certain cultural considerations in mind, we believe these changes can actually improve everyone's experience.

This section discusses some technical strategies to introduce change that may be useful even in the most difficult cultures. You may not be the CEO or a leader in your organization, but every developer, SRE, and security professional is an instrument of change in their own sphere of influence. By making conscious choices about how you design, implement, and maintain systems, it is possible to have a positive effect on your organization's culture; by choosing certain strategies, you may find that over time you can turn the tide by building trust and goodwill.

The advice we give in this section is based on that goal—but culture is something developed over a long period of time, and it's highly dependent on the people and situations involved. When trying out some of the strategies outlined here in your organization, you may find they meet with only limited success, and some strategies may not work at all. Some cultures are static and resistant to change. Yet having a healthy culture that values security and reliability can be just as important as how you design, implement, and maintain your systems, so the fact that your efforts might not always work shouldn't stop you from trying.

Align Project Goals and Participant Incentives

It takes hard work to build trust, but it's quite easy to lose it. In order for people who design, implement and maintain systems to collaborate across multiple roles, they need to share a common reward system.

On a technical level, the reliability and safety of a project can be evaluated regularly through observable metrics like SLOs and threat modeling (for examples, see Chapters 2 and 14). On a process and people level, you should make sure that career advancement opportunities reward security and reliability. Ideally, individuals should be evaluated according to high-level documented expectations. These shouldn't be just a set of checkboxes to tick—they should highlight themes and goals that individuals should aspire to meet. For example, the entry-level Google software engineering job ladder specifies that engineers should master at least one common skill outside of core coding, like adding monitoring to their services or writing security tests.

Aligning project goals with your organization's strategy without aligning participant incentives might result in an unfriendly culture, whereby the people focused on improving the security and reliability of your products are not the ones who tend to get promoted. Since financial rewards regularly correlate to seniority, it's only fair to keep the employees who contribute to happy users happy by aligning project incentives to the reward system.

Reduce Fear with Risk-Reduction Mechanisms

Have you ever found yourself wanting to make a significant change, such as a rollout of new software or a new control, only to find that the organization pushes back because of the perceived risk? You can inspire confidence in your organization by making good deployment choices. We discuss many of these concepts Chapter 7, but it's worth specifically noting the impact of culture here. Here are some strategies you might want to try:

Canaries and staged rollouts

You can reduce fear by slowly rolling out substantial changes through small canary groups of users or systems. That way, the blast radius of an ill-fated change is small if something goes wrong. Also consider going one step further, and implementing all changes via staged rollouts and canaries (see Chapter 16 in the SRE workbook). In practice, this approach has numerous benefits. For example, in Chapter 19 we discuss how the staged release cycle for Chrome balances the competing needs of speedy updates and reliability. Over time, Chrome's staged releases have fostered its reputation as a secure browser. We've also found that by making staged rollouts part of a routine change process, over time, an organization comes to expect that care and diligence are applied to all changes—which builds confidence in change and reduces fear.

Dogfood
: By showing users that you're not afraid of your own changes, you can instill confidence in the stability and productivity impact of any particular change. *Dogfooding* (or "eating your own dogfood") involves adopting a change before that change affects others. This is especially important if you're affecting the systems and processes that impact people's daily lives. For example, if you're rolling out a new least privilege mechanism such as multi-factor authorization, adopt the stricter control within your own team before you require all employees to implement the change. At Google, before we roll out new endpoint security software, we test it on some of our most discerning users (the security team) first.

Trusted testers
: Inviting people in your organization to help test a change early in a project's lifecycle can reduce fear of future change. This approach lets stakeholders see a change before it becomes final, which allows them to raise concerns early. These newly open lines of communication give them a direct channel to deliver feedback if something goes wrong. Showing a willingness to gather feedback during testing phases can reduce silos between parts of your organization. It's important to make clear to the testers that you trust the feedback they're giving, and to make use of their feedback so they know they're heard. You can't always address every piece of feedback—not all of it will be valid or actionable—but by explaining your decisions to your tester population, you can build a strong coalition of trust.

Opt in before mandatory
: A corollary to the dogfood and trusted tester strategies is making a new control optional before it becomes mandatory. This gives teams the opportunity to adopt changes on their own timeline. Complicated changes, such as new authorization controls or testing frameworks, have a cost; it can take time for an organization to fully adopt such changes, and you often need to balance these changes against other priorities. If teams know they have time to implement changes at their own pace, they may be less resistant to doing so.

Progressive stringency
: If you have to put into effect a strict new policy for reliability or security, consider whether you can ratchet up the stringency over time: perhaps you can first introduce a lower-level control that has less impact before teams fully adopt a more stringent one with a heavier burden. For example, suppose you want to add least privilege controls that require employees to justify their access to certain data. Users who don't justify the access appropriately will be locked out of the system. In this scenario, you could start by having the developer team integrate the justification framework (such as a library) into the system, but keep end user justifications optional. Once you deem the system to be performant and secure, you could require justifications to access data without locking out users who fail

to meet the established criteria. Instead, the system could provide detailed error messages when a user enters an inaccurate justification, providing a feedback loop to train users and improve use of the system. After a period of time, when metrics show a high success rate for proper justifications, you can make the stringent control that locks users out of the system mandatory.

Make Safety Nets the Norm

Reliability and security improvements often require you to remove long-relied-upon resources that don't measure up to the new safety standards you're introducing. For example, imagine you want to change how people in your organization use Unix root privileges (or similar highly privileged access), perhaps by implementing new proxy systems (see Chapter 3). Fear of substantial changes like these is natural. After all, what if a team suddenly loses access to a resource that's mission-critical? What if the change results in downtime?

You can reduce fear of change by providing safety nets like breakglass procedures (discussed in Chapter 5) that allow users to bypass a new stringent control. These emergency procedures should be used sparingly, however, and subjected to a high level of audit; they should be viewed as a last resort, not a convenient alternative. When implemented properly, breakglass procedures can provide nervous teams with the assurance that they can adopt a change or react to an incident without completely losing control or productivity. For example, suppose you have a staged rollout procedure that requires a long canary process, which you've implemented as a safety mechanism to prevent reliability issues. You can provide a breakglass bypass mechanism to make the push happen immediately if absolutely necessary. We discuss these types of situations in Chapter 14.

Increase Productivity and Usability

A fear of increased friction can make organizational changes with regard to security and reliability difficult. If people view new controls that slow down development and innovation as counterproductive, they may assume that their adoption will have a negative impact on the organization. For this reason, it's often important to think carefully about the adoption strategy for new initiatives: consider the amount of time necessary to incorporate the change, whether the change might slow down productivity, and whether the benefit outweighs the cost of making the change. We've found that the following techniques help decrease friction:

Build transparent functionality
: In Chapters 6 and 12, we discuss relieving developers of the responsibility for security and reliability by using secure-by-construction APIs, frameworks, and libraries. Making the secure choice the default choice helps developers do the right thing without placing a heavy burden on them. This approach reduces

friction over time because developers not only see the benefits of having secure and reliable systems, but also recognize your intent to keep these initiatives simple and easy. We've found this can build trust between teams over time.

Focus on usability

A focus on usability positively affects a culture of security and reliability.[12] If a new control is easier to use than what it's replacing, it can create positive incentives for change.

In Chapter 7, we talk about how we focused on usability when rolling out security keys for two-factor authentication. Users found that touching a security key to authenticate was much easier than typing in one-time passwords generated by hardware tokens.

As an added bonus, the enhanced security of these keys allowed us to require less frequent password changes.[13] We performed a risk analysis on this topic, considering tradeoffs in usability, security, and auditability. We found that security keys negate the efficacy of password theft by a remote attacker. When combined with monitoring to detect suspected compromise of passwords,[14] and enforcement of password changes in such an event, we were able to balance security and usability.

There are other opportunities where security and reliability features can deprecate old or unwanted processes and increase usability. Taking advantage of these opportunities can build user confidence and trust in security and reliability solutions.

Self-registration and self-resolution

Self-registration and self-resolution portals empower developers and end users to address security and reliability issues directly, without gating on a central team that may be overloaded or slow. For example, Google uses deny and allow lists to control which applications can run on the systems that employees use. This

12 Usable solutions for security and privacy have long been recognized as key to successful deployment of technology controls. For a flavor of what these conversations look like, you might be interested in exploring the proceedings of SOUPS, a conference dedicated to usable security and privacy.

13 Studies have shown that users make poor choices that put their passwords at risk. For more information on the adverse effects of user password choices, see Zhang, Yinqian, Fabian Monrose, and Michael K. Reiter. 2010. "The Security of Modern Password Expiration: An Algorithmic Framework and Empirical Analysis." *Proceedings of the 17th ACM Conference on Computer and Communications Security*: 176–186. *https://oreil.ly/NbfFj*. Standards and compliance regimes are also considering these effects. For example, NIST 800-63 has been updated to require a password change only when there is suspicion that it has been compromised.

14 Password Alert is a Chrome browser extension that alerts when a user has typed their Google or GSuite password into a malicious website.

technology is effective in preventing execution of malicious software (such as viruses).

The downside is that if an employee wants to run software not already on the allow list, they need to seek approval. To reduce friction for exception requests, we developed a self-help portal called Upvote that enables users to get approval for acceptable software quickly. In some cases, we can automatically determine a piece of software to be safe and approve it. If we can't automatically approve the software, we give the user an option to have it approved by a set number of peers.

We've found social voting to be a satisfactory control. It's not perfect—sometimes employees approve software that's not necessarily business-related, such as video games—but this approach has had a high rate of effectiveness in preventing malware from being executed on our systems. And since it does not gate on a central team, friction for the control is kept very low.

Overcommunicate and Be Transparent

When advocating for change, the means of communication can influence outcomes. As we discuss in Chapters 7 and 19, good communication is key to building buy-in and confidence in success. Giving people information and clear insight into how change is happening can reduce fear and build trust. We've found the following strategies to be successful:

Document decisions

When making a change, clearly document why it's happening, what success looks like, how the change will be rolled back if operating conditions deteriorate, and who to talk to in case of concerns. Make sure that you clearly communicate why you're making the change, especially if it directly affects employees. For example, every Production Excellence SLO at Google requires a documented rationale. Since the SRE organization is measured against these SLOs, it's important that SREs understand the meaning behind them.[15]

Create feedback channels

Make communication bidirectional by creating feedback channels through which people can raise concerns. This could be a feedback form, a link to your bug tracking system, or even a simple email address. As we mention in the discussion of trusted testers (see "Reduce Fear with Risk-Reduction Mechanisms"

15 Production Excellence reviews are carried out periodically on SRE teams by senior SRE leaders, assessing them on a number of standard measures and providing feedback and encouragement. An SLO of 99.95% might be accompanied by a rationale such as, "We previously wanted to reach a 99.99% success rate, but found this target to be unrealistic in practice. We have not discovered a negative impact on developer productivity at 99.95%."

on page 484), giving partners and stakeholders a more direct involvement in a change can lessen fear.

Use dashboards
: If you're making a complex change across multiple teams or parts of the infrastructure, use dashboards to show clear expectations of what you need people to do and provide feedback on how well they're doing. Dashboards are also helpful in showing the big picture of a rollout and keeping the organization in sync on progress.

Write frequent updates
: If a change takes a long time (some changes at Google have taken several years), assign someone to write frequent (for example, monthly) stakeholder updates outlining progress. This will build confidence—especially in leadership—that the project is progressing and that someone has a watchful eye on the health of the program.

Build Empathy

> *You can't understand someone until you've walked a mile in their shoes.*
> *—Unknown*

People begin to understand the challenges others face when they understand the ins and out of performing their role. Cross-team empathy is especially important when it comes to reliability and security properties of the system, since (as discussed in Chapter 20) these responsibilities should be shared across the organization. Building empathy and understanding can help reduce fear in the face of necessary changes.

In Chapter 19, we outline a few techniques for building cross-team empathy—in particular, how teams can share responsibilities for writing, debugging, and fixing code. Similarly, the Chrome security team runs fixits not only to improve the security of the product, but also as a cross-organization team-building activity. Ideally, teams consistently share responsibilities from square one.

Job shadowing or job swapping is another approach to building empathy that doesn't require permanent organizational top-down changes. These engagements can range from a few hours (which tend to be less formal exercises) to several months (which may require management buy-in). By inviting others to experience your team's work, you can signal that you're willing to tear down organizational silos and build common understanding.

Google's SRE Security Exchange Program allows an SRE to shadow another SRE or security engineer for a week. At the end of the exchange, the SRE writes a report with improvement recommendations for both their home team and the host team. When conducted in the same office, this program requires a very low investment but

provides many benefits in terms of knowledge sharing across the organization. Google's Mission Control program encourages people to join the SRE organization for six months, during which time they learn how to think like an SRE and respond to emergencies. In doing so, they directly see the impact of software changes initiated in partner organizations. A parallel program known as Hacker Camp encourages people to join the security team for six months, where they can work on security reviews and response efforts.

Programs like these may begin as small experiments with one or two engineers, and grow over time if successful. We've found that this type of job swapping both builds empathy and inspires exciting new ideas about how to solve challenges. Bringing in these new perspectives and building goodwill between teams helps grease the cogs of change.

Finally, building in mechanisms to say thank you—from simple emails to more elaborate forms—reinforces the positive impact that people have on one another and sets the right incentives. At Google, we've long had a culture of peer bonuses—small amounts of cash that don't cost the company a lot of money, but build large amounts of goodwill. A cash-free version of this called Kudos allows Googlers to formally recognize each other in digital form that's visible to everyone. Some of our offices have also experimented with thank-you postcards.

Convincing Leadership

If you work in a large organization, getting buy-in for reliability and security changes you want to make may be a challenge. Since many organizations are incentivized to spend their limited resources on revenue-generating or mission-forward efforts, it can be tough to get buy-in for improvements that are seen as happening behind the scenes.

This section explores some strategies we've used at Google, or seen used elsewhere, to get buy-in from leadership for security and reliability changes. As with the guidance given elsewhere in this chapter, your mileage may vary. Some of these strategies will be effective, while others won't be. Just as every organization's culture is unique, so is every leader and leadership team. It's worth repeating our previous advice here: just because you think one of these strategies won't work doesn't necessarily mean you shouldn't try it. The results just might surprise you.

Understand the Decision-Making Process

Suppose you want to make a fairly substantial change to your organization's custom frontend web-serving infrastructure to include DDoS protection; for example, referencing the benefits outlined in Chapter 10. You know this will vastly improve the reliability and security of the system, but it also requires multiple teams to

incorporate new libraries or restructure code. Integrating and testing this change properly could take months. Given the high cost but positive impact, who in your organization would make the decision to move forward, and how would they make that decision? Understanding the answers to these questions is key to knowing how to influence leadership.

Here, the term *leadership* loosely applies to the people who make decisions, whether those decisions are around direction setting, resource allocation, or resolving conflicts. In short, these are the people who are perceived to have authority and accountability. They are the people you want to influence, so you need to figure out who they are. If you work in a large company, they could be VPs or other senior people in management. Smaller organizations, such as startups and nonprofits, often consider the CEO to be the senior decider. In an open source project, this could be the project's founder or top contributor.

The answer to the question "Who is the decider for this change?" can be tricky to determine. The authority to make decisions may indeed lie with someone typically considered to be at the top of the leadership hierarchy or in an obvious gatekeeping role, such as a lawyer or risk officer. But depending on the nature of the change you're proposing, the decision might also reside in a tech lead, or even with you. The decider may not be a single person; it could be a set of stakeholders across the organization from different departments such as legal, press relations, engineering, and product development.

Sometimes the authority to make a decision resides loosely within a group of people or, in the extreme form, within a whole community. For example, in Chapter 7 we describe how the Chrome team participated in increasing HTTPS usage on the internet. In this situation, the decision to make a directional change was made within the community, and required building an industry-wide consensus.

Determining who is a decider for a change may take some sleuthing, especially if you are new to an organization, or if there are no existing processes telling you how to get something done. However, you can't skip this step. Once you understand who the deciders are, you then should seek to understand the pressures and demands that they face. These might stem from their own management, boards of directors, or shareholders, and can even take the form of external influences like customer expectations. It's important to understand these pressures so you can understand where your proposed changes fit in. To return to our earlier example of adding DDoS protection to the frontend web-serving infrastructure, where would that change fit in relative to leadership's priorities?

Build a Case for Change

As we've mentioned already in this chapter, resistance to change can stem from fear or perception of friction, but in many cases it can also stem from not understanding

the reason for a change. When faced with many priorities, decision makers and stakeholders have the difficult task of choosing between different goals that they would like to achieve. How will they know your change is valuable? It's important to understand the challenges decision makers face when building a case for your change. These are some of the steps in a successful case-building process:[16]

Gather data
: You know that a change needs to be made. How did you come to that conclusion? It's important to have data to back up your proposed change. For example, if you know that building automated test frameworks into the build process will save developers time, can you demonstrate how much time this change will save? If you're advocating for continuous builds because the practice creates incentives for developers to fix errors, can you show how continuous builds save time in the release processes? Conduct research and user studies to produce data-rich reports complete with graphs, charts, and anecdotal stories from users; then summarize this data in a way that decision makers can digest. For example, if you want to drive down the time it takes your team to patch security vulnerabilities or address reliability configuration issues, consider creating a dashboard that tracks progress for each of the engineering teams. Showing those dashboards to the leaders of those areas can encourage individual teams to hit targets. Be mindful of the investments you'll need to make to gather high-quality, relevant data.

Educate others
: Security and reliability issues can be hard to understand unless you're connected to them every day. Get the word out through talks and information sessions. At Google, we use Red Team postmortems (see Chapter 20) to educate leaders at a high level about the kinds of risks we're facing. While Red Teams were not originally created as an educational effort, they can raise awareness within all levels of the company. This has been beneficial in convincing teams to maintain their SLOs for remediating vulnerabilities.

Align incentives
: Using the data you've gathered and your knowledge of the pressures deciders face, you may be able to address other concerns in their sphere of influence. In our earlier DDoS example, making the proposed change to the framework would provide a security benefit, but a more reliable website would also potentially help increase sales. This could be a strong argument to present to the company's leadership. For a real-world example, Chapter 19 discusses how rapid releases of Chrome get security fixes to users faster, with the additional benefit of quick deployment for reliability fixes and new features. This is great for users and

16 For a practical example of how we successfully built a case for change, see "Example: Increasing HTTPS usage" on page 136.

product development stakeholders alike. Don't forget to discuss how you're reducing the fear and friction that may accompany a change—as mentioned earlier in this chapter, Google's rollout of security keys allowed us to eliminate unpopular password change policies and reduce end-user friction for two-factor authentication, which were powerful arguments for change.

Find allies
: Chances are, you're not the only person who knows that a change you're proposing would be beneficial. Finding allies and convincing them to support your change can add weight to your argument, especially if those people are organizationally close to the decision makers. Allies can also test your assumptions about a change. Perhaps they know of different data-based arguments, or understand the organization in a way that you don't. This type of peer review can bolster the strength of your argument.

Observe industry trends
: If you're adopting a change that other organizations have already adopted, you may be able to rely on their experiences to convince your leadership. Do your research—articles, books, public talks at conferences, and other materials may demonstrate how and why an organization took up a change. There may be additional data points you can use directly in building your case for change. You could even consider bringing in expert speakers to address your leadership on specific topics and industry trends.

Change the zeitgeist
: If you can change the way people think about your problem over time, it may be easier to convince decision makers later on. This applies especially when you need broad consensus for a change. We discuss this dynamic briefly in the HTTPS case study in Chapter 7, where the Chrome team and others in the industry changed developer behavior over a long period of time, to the point where HTTPS as a default became the norm.

Pick Your Battles

If your organization is facing many reliability and security challenges, constant advocacy can create fatigue and resistance to additional change. It's important to pick your battles carefully: prioritize initiatives that have a chance of succeeding and know when to stop advocating for lost causes. This shows leadership and decision makers that you're tackling the most important issues.

Lost causes—that is, proposals you have to shelve—have value too. Even when you can't successfully advocate for change, having data that supports your ideas and allies that back your plan, and educating people about the problem, can be valuable. At some point, your organization may be ready to tackle a challenge you've already studied. If you already have a plan waiting in the wings, teams can move faster.

Escalations and Problem Resolution

Despite best efforts, sometimes the need to make decisions on a security or reliability change can rise to the surface in an explosive way. Perhaps a serious outage or security breach means that you quickly need more resources and staffing. Or perhaps two teams have differing opinions on how to solve a problem, and the natural course of decision making isn't working. In these types of situations, you may need to seek resolution from the management chain. When dealing with escalations, we recommend the following guidelines:

- Form a group of colleagues, mentors, tech leads, or managers to provide input on the situation from both sides. It's usually a good idea to walk through the situation with someone with an unbiased view before deciding to escalate.
- Have the group summarize the situation and proposed decision options for management. Keep this summary as concise as possible. Maintain a strictly factual tone, and include links to any relevant supporting data, conversations, bugs, designs, etc. Make the potential impact of each option as clear as possible.
- Share the summary with your own team's leadership to ensure further alignment on possible solutions. For example, multiple issues might require simultaneous escalation. You may want to either merge escalations or emphasize other aspects of corresponding situations.
- Schedule a session to present the situation to all affected management chains and designate appropriate decision makers in each chain. The decision makers should then make a formal decision or meet separately to discuss the issue.

As a concrete example, sometimes security issues at Google need to be escalated when an unresolvable disagreement arises between the product team and the security reviewer about the best course of action. In this case, an escalation is initiated within the security team. At that point, the two senior leaders within the organizations negotiate a compromise or decide to implement one of the options suggested by the security team or the product team. Because we integrate these escalations into our normal company culture, escalations aren't seen as confrontational.

Conclusion

Just as you design and manage systems, you can design, implement, and maintain the culture of an organization over time to support security and reliability goals. Reliability and security efforts should be considered just as carefully as engineering efforts. There are important cultural elements of engineering that, when taken in aggregate, or even on their own, can contribute to more robust systems.

Security and reliability improvements can inspire fear or concern about increased friction. There are strategies for addressing these fears and helping achieve buy-in

from the people these changes affect. Making sure that your goals are well aligned with stakeholders—including leadership—is key. Focusing on usability and demonstrating empathy for users can encourage people to adopt change more readily. Making a small investment in thinking about how others perceive change may lead to greater success in convincing them that your changes are sound.

As we stated in the opening to this chapter, no two cultures are the same, and you'll need to adapt the strategies we've outlined to your own organization. In doing so, you will also find that you likely can't implement all of these strategies. It may be useful to pick and choose the areas your organization most needs to address, and improve those over time—which is the way Google approaches constant improvement over the long term.