GitHub

Secure software development strategy essentials

Table of Contents

Introduction

Trust is the foundation of the relationship between software companies and their customers. The ability to prevent sensitive data from falling into the wrong hands is a cornerstone of this trust. Globally, governments have responded with a variety of regulatory frameworks obligating companies to protect sensitive customer data, with severe sanctions imposed on firms that fail to do so. Failure to properly safeguard data can have profound and lasting impact on a company's reputation and their bottom line.

To make matters more complex, software has become a central component of businesses everywhere and will continue to increase in importance. However, strict regulatory and compliance requirements can govern the way teams build software, from the processes they follow to the tools they use. These requirements are often prohibitive enough to compromise performance and stifle innovation, particularly for companies in highly regulated industries, such as healthcare, government, and financial services.

The good news is secure development doesn't have to be a barrier to collaboration or innovation. Businesses today are uniquely positioned to create impactful, forward-thinking user experiences, and many of them are doing just that. Thousands of organizations are using GitHub to free their workflows from insular development and build secure processes that give engineering teams the flexibility to do their best work.

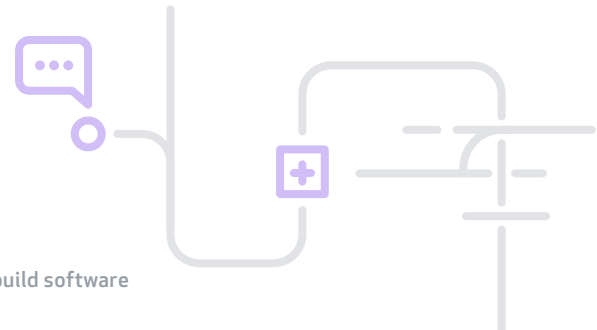
With consumer expectations higher than ever and increased pressure to lower costs, efficient, collaborative, and secure workflows can help teams shift focus to where it matters most: Building the best, most innovative software for their customers.

In this guide, we'll outline the unique regulatory and technical challenges that software companies face, how to address them, and how GitHub can help.

Safeguarding customer data

Today, customers share some of their most sensitive data with companies in online transactions and other data exchanges. Security teams are engaged in a constant arms race with those trying to obtain this data for criminal purposes—from individual hackers to state actors deploying considerable skill and resources in their efforts to undermine security. Fortunately, there are best practices and tools that can help companies secure their development processes and keep sensitive data safe.

There is no single solution or approach that can guarantee security. Instead, effective security relies on a layered approach, enforcing checks and safeguards at multiple points across data paths and workflows. This layered approach includes a combination of tools, practices, and culture.

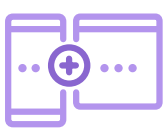


Build a culture of security

A corporate culture that places security at the center of everything is key to enforcing security policies at scale. Creating a security culture means building processes that make everyone's job. Examples include regular training on secure practices, highlighting the danger of social paths to infiltration like phishing attacks, as well as technical means. Specific best practices include:

SPECIFIC BEST PRACTICES INCLUDE:

Enforcement of password policies, including complex passwords, regular password changes, and the use of password managers and generators to enhance password security



Use of two-factor authentication

Education and training regarding the dangers of public networks, as well as the physical protection of devices like laptops, tablets, and cell phones.

Use of VPNs to secure access to internal corporate networks

Always encrypt data

One of the best ways to avoid unauthorized access to data is encryption. Even if a bad actor obtains access to data transmission, strong encryption can render that success of little or no value. Again, employing a layered approach is most effective when implementing an encryption strategy. This means:

Encrypting data in transit when it's being transmitted between a permanent store and an end user

Encrypting data 'at rest' in the databases where it's stored

ENCRYPTION IN TRANSIT

As a standard practice, any web-based effort should employ end-to-end encryption using modern encryption protocols such as TLS. TLS helps secure communication using strong, certificate-based encryption that can be essentially unbreakable in any practical sense. TLS also verifies each party in a network transaction, preventing 'man-in-the-middle' attacks, where a bad actor poses as a trusted partner in a transaction. Services like LetsEncrypt have greatly reduced the cost and complexity of implementing effective TLS and obtaining the required certificates.

ENCRYPTION AT REST

Some applications provide encryption services at the software layer. This sort of encryption is better than none but can bring significant costs in terms of complexity and reduced performance. Encryption solutions that run as close to the storage hardware as possible, such as filesystem encryption or hardware appliance level encryption can provide a transparent layer of encryption for all applications.

ENCRYPTION WITH GITHUB

All communication to and from our self-hosted solution GitHub Enterprise Server is protected by TLS with the option to use LetsEncrypt for inexpensive and simple certificate maintenance. GitHub also secures communication between Git clients used by developers and the server using SSH. Both are strong cryptographic protocols that present robust barriers to unauthorized interception or leakage of sensitive data.

Adopt strong identity management practices

The more passwords users have to remember, the more likely they are to default to insecure practices like writing passwords down or reusing easy-to-remember (and thus easy to crack) passwords. Using a centralized identity management system like LDAP coupled with a Single Sign On (SSO) solution can greatly reduce the number of passwords users have to manage, making it easier for them to follow sound password management practices.

USING GITHUB WITH IDENTITY MANAGEMENT SYSTEMS

GitHub Enterprise supports authentication and authorization using a variety of identity management solutions like Active Directory and LDAP. GitHub also supports SAML, which allows enterprises to provide a Single Sign On experience for users.

Prevent database attacks

Data entering an organization can be a dangerous vector that bad actors use to attempt to gain access to your systems. Many of these strategies rely on passing dangerous payloads to poorly designed systems, exploiting flaws to gain control. SQL injection is a classic example, occuring when a bad actor intentionally appends SQL code to seemingly harmless data like a customer name in a web application. Flaws in the underlying software can result in arbitrary execution of this code, resulting in data being unintentionally returned to the user. These attacks are particularly dangerous because they may not cause an error or other event that might attract the attention of IT administrators overseeing security.

Again, a layered approach is most effective in ensuring that only valid data enters your environment. This begins with the code in the user interface and extends throughout the datapath through middleware to underlying data stores. Building validation into the code at each step can dramatically reduce your vulnerability to these sorts of attacks. A combination of frequent manual and automated reviews and tests are critical to supporting this effort.

Three ways you can prevent common vulnerabilities:

- Only accept validated data into production systems
- Monitor systems for errors and edge cases
- Restrict access to data systems of records as much as possible by adopting an identity management system

Segregate sensitive data

The fewer places sensitive data exists, the fewer opportunities (or less "surface area") attackers have available to exploit. Reducing surface area means implementing practices like storing passwords in a password manager rather than entrusting them to individuals and never committing passwords, access/API tokens, encryption keys, and other sensitive data to publicly accessible repositories.

Sensitive data should *never* be used or stored in non-production systems. Of course, teams might require large amounts of data that reliably mimic the real thing for testing, tempting some engineers to 'refresh' development and testing environments with production data. Instead of using production data, consider using a tool that allows you to configure and then generate large sets of dummy data. These allow reliable, realistic testing without risking sensitive information.

PREVENTING SENSITIVE DATA FROM REACHING REPOSITORIES WITH PRE-RECEIVE HOOKS

In addition to storing sensitive data separately, you can prevent the inadvertent (or deliberate) committing of protected data to repositories with **pre-receive hooks**—simple scripts that run in an isolated environment on the GitHub appliance. These scripts are triggered _before_ code is pushed to GitHub to examine commits, identify sensitive information, and prevent it from being added to your repositories.

After examining code, pre-receive hooks return only one of two possible values: success, or failure. If it fails, developers will see a message informing them that their commit failed, along with any other useful information your team chooses to include. The code is committed to the repository only if the pre-receive hooks succeeds. The result? No unwanted code makes it to your repositories, protecting your company from violations, liability threats, and potential regulatory penalties.

Enforce protections on source code

Effective security requires controlling access to sensitive information and the source code for the software that manages it. Granular controls mean you can effectively protect this information without creating a security-bound environment that denies access too comprehensively. GitHub can support your access policies with **protected branches**. Protected branches help maintain the integrity of your code by limiting several features of Git on any branch an administrator chooses to protect. For example, administrators can restrict who can post to a branch to specific users and teams programmatically. They can also disable force pushes that might change or delete code on certain branches. Additional safeguards include:

ADDITIONAL SAFEGUARDS INCLUDE:

Preventing deletion of a branch

Requiring manual, auditable reviews from one or more people

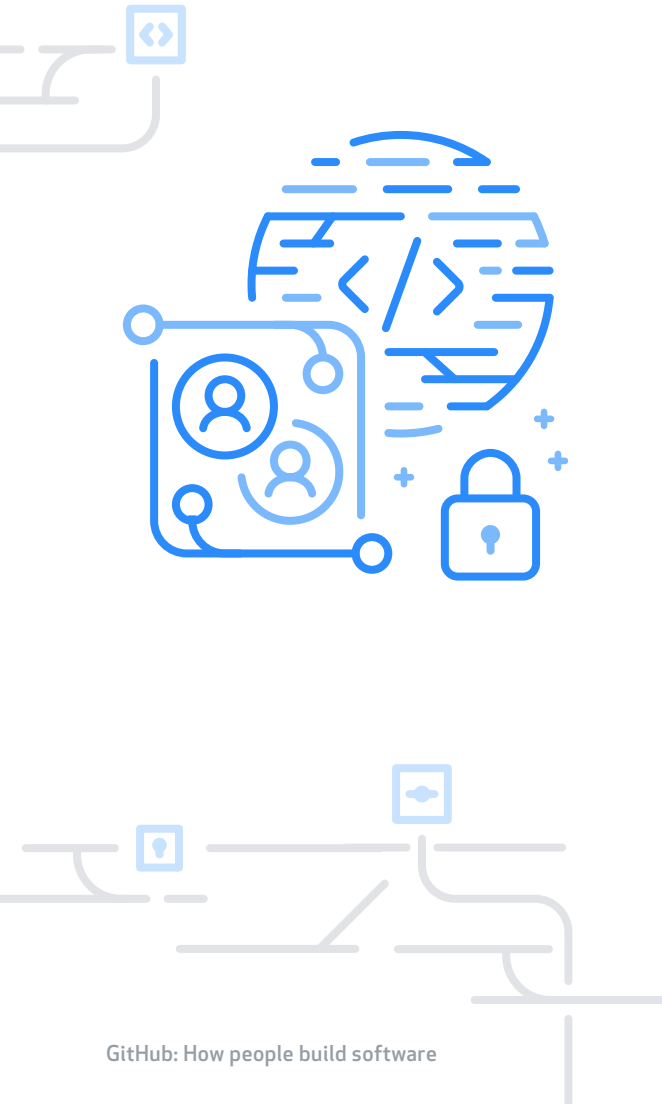
Preventing code which fails automated testing and Continuous Integration (CI) checks from being merged

Specifying 'codeowners' for any part of a codebase and requiring their review

Steps you can take to protect your customer's most important information:

- Use secure connections everywhere (no excuses!) and only transfer data using protocols like SSH and SFTP
- Never commit passwords, access/API tokens, encryption keys, and more to publicly accessible repositories.
- Use protected branches and pre-receive hooks to account for human error and protect sensitive data from making it to production
- Use an identity solution to restrict access and make sure only the right people have access to sensitive data

Securing the entire development process



Your software codebase and the development processes that shape the way your team builds are at the center of a layered security strategy. As developers work together and contribute changes, it's important to put certain safeguards in place through a combination of best practices and GitHub features that make sure the code that reaches production is secure.

Include legacy core systems

Tools like mainframes remain central to some companies' operations more than 50 years after their initial introduction. These systems provide unparalleled capabilities but can also pose unique challenges to secure. Development teams should periodically evaluate all legacy systems and balance the cost of updating them against the potential risks of causing a security breach. Replacing a legacy system may not seem cost-effective until the threat of fines in the billions of dollars or a devastating security breach destroying public goodwill are included in the equation.

SELECTING A MAINFRAME SYSTEM

GitHub's partner ecosystem includes many vendors producing, selling, and maintaining mainframe systems. For example, IBM recently announced extensions for Git, the technology underpinning GitHub, that allow modern DevSecOps practices to be applied to mainframe development workflows. Find integrations to power your GitHub.com workflow on GitHub Marketplace or learn more about which apps work with our self-hosted solution GitHub Enterprise Server at github.com/works-with.

Automate processes to prevent bugs and errors

organizations that implemented automated dependency management, for example, had **60 percent** fewer security vulnerabilities in their delivered software than those which did not.

Manual oversight is a critical component of an effective, layered security strategy. However, overreliance on people can be dangerous. People get tired, become distracted, or simply make mistakes. Put people to work where their skills are most important, and delegate repetitive, tedious, yet still critical tasks to machines. Your team can leverage fully automated **Continuous Integration and Delivery (CI/CD)** in your GitHub workflows.

CI/CD tools test and evaluate your code every single time a commit is pushed to a repository. With CI/CD in place, GitHub can test new code with existing production code to ensure the proposed changes work as intended and do not introduce security flaws into existing systems. These tests can also extend to examining all the code that your code depends on (its dependencies).

Implementing automation isn't difficult—and it can yield immediate and measurable results. In a 2017 study, the IEEE found that organizations that implemented automated dependency management, for example, had 60 percent fewer security vulnerabilities in their delivered software than those which did not.

INTEGRATING CI/CD WITH GITHUB

Integrating CI/CD is easy with GitHub, and it won't affect developers' existing workflows unless an automated test result requires them to take action. The less intrusive automation is for developers, the more likely it is to be widely adopted and its benefits felt across an organization.

GitHub integrates with a variety of CI tools like Jenkins, Travis, and CircleCI. These automatically fetch code from a GitHub repository every time code is pushed, run tests, and return the results to GitHub with either a pass/fail status. They also return notifications on the status of each test, so developers can see where their code is failing. And with the Checks API, teams can build sophisticated custom tools for CI that make feedback seamless and richly informative.

Learn more about the Checks API

You and your team can decide whether or not you'd like the results of CI tests to prevent code from getting merged into the code base or simply alert developers without taking action. If CI statuses are required, the pull request can only be merged when all required CI jobs complete. If you choose to prevent merging, the pull request can't be merged until the required tests return successfully.

Integrating security tools and workflows

Hundreds of tools integrate with GitHub—and many of them can help you keep your code and customer data safe.

- **CI:** CI tools like Travis CI, CircleCI, and AppVeyor automatically build and test code as you push it to GitHub, preventing bugs from being deployed to production
- **The Checks API:** Teams can create sophisticated custom tools that report on exactly the data and feedback you need
- **Error reporting:** Tools like Snyk, Sentry, and Dependabot help your team find, fix, and prevent vulnerabilities
- **Code quality:** Code Climate and Codacy automate reviews with security and quality checks to prevent human error and streamline your team's code review processes

Ready to see what kinds of tools are available to your team? Visit GitHub Marketplace at github.com/marketplace or browse github.com/works-with

Enforce manual reviews with required code review

Automation is important, but manual review of code will always remain a critical component of your security strategy. The more eyes on a given codebase, the more likely errors or vulnerabilities are likely to be detected. Reviews also serve a valuable function beyond security, helping ensure that institutional knowledge is shared and providing learning and mentoring opportunities for developers of all skill levels. The result of regular, organization-sanctioned reviews is a codebase that is not only more secure but also healthier and more consistent.

GitHub provides a flexible framework for mandating code reviews. You can:

Require reviews from one or more users with write access to the repository containing the changes

Empower reviewers to provide detailed, line-by-line feedback on any proposed changes

Require additional review and approval by one or more **codeowners**. Codeowners can be any combination of individual users or teams assigned specific responsibility for a section of a codebase, a particular technology, or any combination of the two

Ensure that all reviews are logged and auditable in the future

Three steps to take to help your team prevent bugs and errors:

1. Build in automated security scanning to reduce human error, test for weaknesses that may not be introduced otherwise, and save developers time
2. Integrate automation directly into your team's GitHub workflow with status checks
3. Ensure all of your team members feel an equal and shared responsibility for building and maintaining secure software

Audit and compliance-proof your workflow

The regulatory standards under which some companies work today require a robust logging and auditing capability. Not only do teams need to create an effective security strategy, they also have to prove they've done so to regulators. GitHub provides flexible and powerful logging, auditing, and reporting frameworks to help ensure compliance and the ability to prove it.

AUDIT LOGGING:

GitHub Enterprise maintains comprehensive logs of user and system activity. Audited activity includes every `git push` operation, including who initiated the push, their IP address, and repositories affected by the operation. Logs can be forwarded to an external system (like Logstash or Splunk) for analysis, reporting, and storage, ensuring compliance with regulatory requirements

ARCHIVING REPOSITORIES:

Repositories that are no longer maintained can be archived and set to 'read-only' mode. This ensures that no new code can be committed to the repository while still allowing users to view its content

BLOCKING FORCE PUSHES:

Git allows developers to rewrite commit history by 'force pushing' changes. While sometimes necessary in order to fix mistakes, this capability presents a challenge when regulations require data to be immutable. GitHub Enterprise allows administrators to block force pushes for individual repositories or for the entire appliance, ensuring commit history can never be rewritten

PREVENTING USERS FROM DELETING REPOSITORIES:

The ability to delete repositories can be restricted to administrators

Move security all the way left

In the past, security was all too often an afterthought. In fact, the protocols which run the internet were not designed with security in mind _at all_. Bolted-on aftermarket security is better than nothing, but far from sufficient in today's frenzied threat environment. To be effective, security must be built into every software project from its very inception. Moving security left means shifting security from the right-hand side of your timeline, near delivery, all the way to the left, at the beginning. This ensures that security remains front and center throughout the entire product lifecycle.

Involving security teams early can also influence design, development, and maintenance decisions before they become too difficult and expensive to change. This advice is simple enough, but process means very little if leadership and engineering teams aren't on board. Secure development requires everyone to adopt an effective security culture.

CREATING A SECURITY CULTURE

DevSecOps is a way of thinking about security throughout the development cycle and distributing it across teams and roles. It no longer makes sense for security teams to join the process after an application is built, only to discover exploitable flaws. With security experts closely aligned from the start, teams can create collaborative processes that proactively support security as they build.

DevSecOps principles may require organizations to shift their team culture in addition to their infrastructure, but more reliable software and fewer surprises are worthwhile results. Some companies even invite security specialists into scrum teams to make sure it's a priority throughout the process.

Customer story: Anaplan

Anaplan helps businesses make smart, data-driven decisions with advanced modeling software. With a focus on speed, flexibility, and integrity, Anaplan supports planning at more than 500 companies—helping them stay nimble, discover opportunities, and plan for their futures.

Behind its smart platform is a huge amount of data. Anaplan's 150+ developers need to create a platform that's both flexible enough to make that data meaningful to businesses across industries, while ensuring that data is secure enough to keep it private. Their customers trust Anaplan to store sensitive financial plans in its own data centers. As a result, the company has rigorous compliance and security requirements in place to ensure the safety and privacy of regulated financial data.

To help their teams build a flexible, forward-thinking platform, Anaplan uses GitHub Enterprise Server.

"Compliance is a very big deal, and we built a lot of tooling to make sure we're bulletproof. All commits are tagged with JIRA IDs that link everything together, including all of our developer tools. GitHub has made a huge impact on compliance, while helping us become more transparent."

JON SANDLES,
Release Manager, Anaplan

At the heart of the Anaplan's build and deployment pipelines are a set of automated tools that leverage the GitHub Enterprise API to regularly query and poll, and produce reports and alerts across all repositories. These keep the product team in tune with what's going on by emailing developers, creating reports in Confluence, and updating JIRA and Slack channels.

Product Managers are informed if code is checked into the wrong repositories against incorrect JIRA projects. Engineers are kept informed if pull requests are left open for too long. Code is scanned continuously by tools like Sonar and Checkmarx; deployment appliances like Chef scripts get the same treatment. Everything that goes to production follows the same pipelines.

At any point in time, especially near production deployment, the release and program management team are fully up to date on codebase compliance and readiness.

Four key features convinced Anaplan that GitHub offered the best solution:

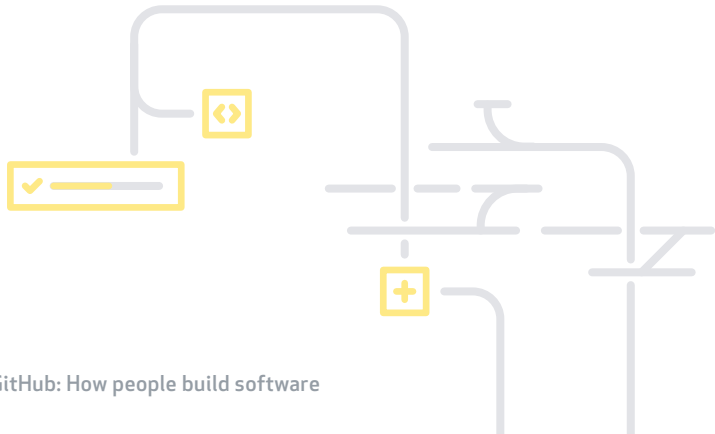
1. **Protected branches:** Developers can ensure code that processes data comes from a known source and gets reviewed
2. **Hard tokens:** Developers can keep access tightly controlled by using two-factor authentication with hard tokens behind a VPN
3. **JIRA integration:** Developers can attach all pull requests and commit comments to JIRA IDs that document whether changes are approved and tested
4. **Documentation versioning:** Programmers can create and host documentation as close as possible to the applications
5. **Security:** the platform is hosted on internal servers; Enterprise Server uses the enterprise directory to manage authentication and logs all user and system activity

Summary

Keeping data safe has never been more critical to the success, even survival of your business. Implementing effective security might seem overwhelming, but it doesn't have to be. One of the advantages of a layered defense is to break up the task of implementing security into multiple, less daunting sub-projects. The most important thing to do is to start, or if you've started, continue to test, examine, question, and improve. Attackers never rest, and neither can you.

Part of the challenge in implementing effective security lies in balancing it with the freedom that software developers need to innovate, thrive, and be productive. Too little security is simply ineffective. Too much, on the other hand, can stifle creativity or encourage people to avoid your security efforts altogether. Tools like GitHub provide you with the controls you need to help find that balance for your organization.

Ultimately our goal is to help you transform your perception of security beyond a threat and into an opportunity to build customer satisfaction, attract new customers, and further differentiate your business. Good security pays off in customer trust—and partners like GitHub can help you on your way to an effective security strategy.



Key takeaways:

1. Your data are under constant attack. A breach can have very serious consequences for your business and, increasingly, for you personally
2. There is no magic fix for security. Instead, a layered defense comprised of multiple tools, processes, and practices is much more resilient and effective
3. Effective security requires an effective security culture. Security must be a stakeholder in every important initiative and decision
4. Encrypt everything—no exceptions
5. Use a multi-layered approach to vet and validate all data entering your systems to prevent malicious attacks like SQL injection
6. Plan for and include older, but still mission-critical systems like mainframes. Include the possible costs in terms of fines, civil liability, and lost goodwill when evaluating and upgrading legacy tools
7. Leverage automation where you can, particularly for software testing

We're here to help

GitHub supports building robust, secure code. Ready to explore how GitHub fits into your secure development process?

Ready to try GitHub? Get started today.

GitHub