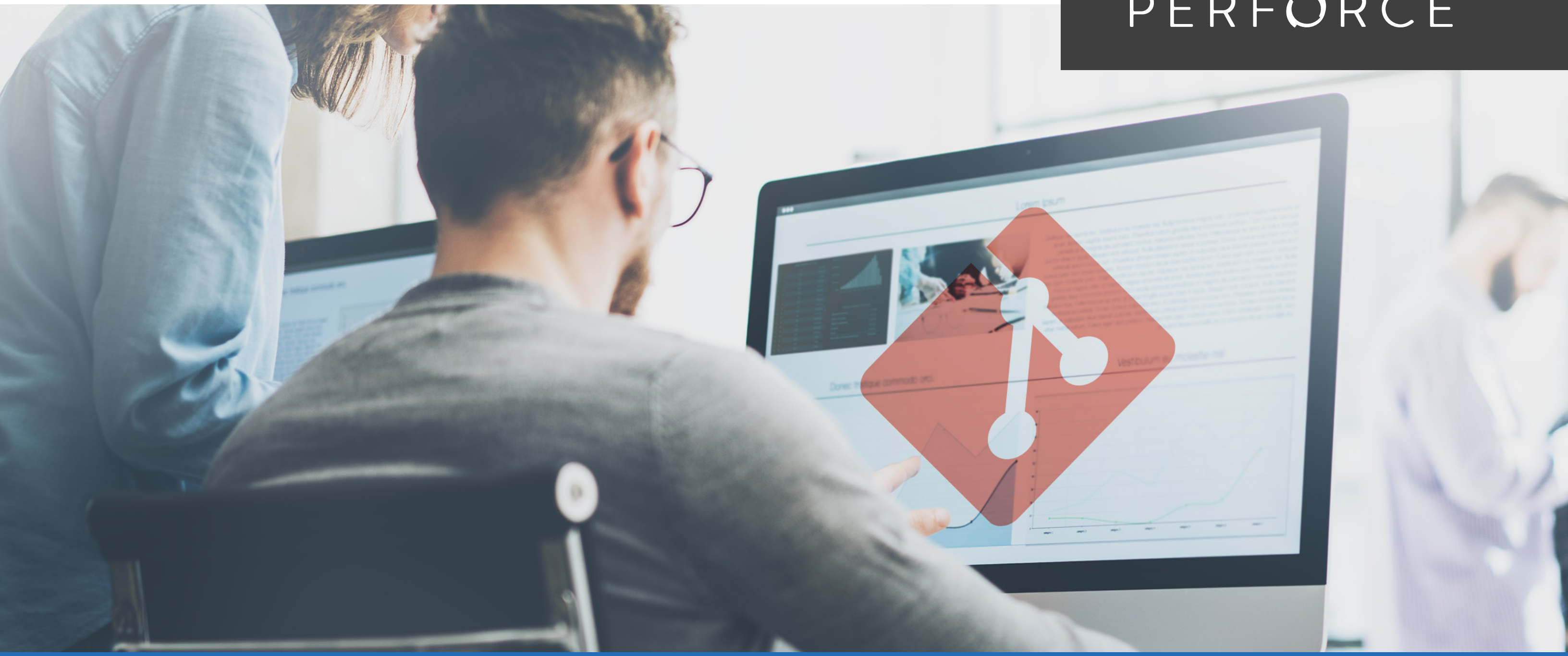

WHITE PAPER

How to Lock Down Git

A CTO's Guide to Avoiding Bad Behavior and Minimizing Git Security Risks

Introduction

Git allows developers to work together efficiently. It also allows bad behavior, unless development leaders do something about it. This can lead to potential loss of intellectual property and significant security issues. This white paper covers how to lock down Git and mitigate these risks.

PERFORCE

Contents

Native Git Lacks Security

There are no security measures in native Git. Developers pull down entire repositories every time they need to make changes to source code. Each time, this exposes code to potential risks. Often, there are no secure backups of Git servers. The only backups are the clones of repositories sitting on developer laptops.

The utter lack of security measures makes using native Git a major risk for company intellectual property (IP).

GIT INVITES BAD BEHAVIOR

With native Git, the responsibility is on each individual to behave well. Developers contribute to and collaborate on code in Git repositories. Each time they work on code, they download the entire repository — with full history — onto their laptops.

It makes it easy for developers to behave badly, whether they intend to or not. That exposes the entire repository to risk.

From Carelessness…

If a careless developer pulls down a repository, they could unwittingly expose data to hackers.

For example, if their workstation is compromised by a phishing attack, they could put code out where it doesn't belong. Another example is that they could clone from clones — and there would be no way to know where the IP went. Or, a developer could be careless while executing a command. For instance, a developer could execute a force-push, which would eliminate other developers' changes and history.

There is not a magical way to secure every developer's workstation effectively to prevent these scenarios from happening.

As a result, the carelessness of an individual developer can be catastrophic.

…To Malicious Intent

If a malicious developer pulls down a repository, they could intentionally put data at risk. In some cases, this could be a developer leaving to work for a competitor — and taking data with them. In other cases, it could be intent to expose data publicly.

In any case, the malicious intent of a developer can compromise IP, as well as the version history which could expose security vulnerabilities in unpatched releases.

BAD BEHAVIOR LEADS TO SECURITY RISKS

Bad behavior puts codebases and IP at risk. Without proper security measures in place, organizations risk cyberattacks, including exfiltrations, infiltrations, and ransomware.

According to the 2018–2019 Global Application and Network Security Report, 93% of organizations experienced cyberattacks over the course of a year — leaving just 7% who hadn't.

Once hackers have infiltrated a system, they can go unnoticed for years. In a 2018 Marriott data breach, it was reported that hackers went unnoticed for four years. This enabled hackers to pull off one of the largest thefts of personal records of all time.

Data stored insecurely is data lost. And using Git can expose organizations to the same sort of security risks — and theft of data (exfiltration)— that Marriott experienced. That is why it is time for development leaders to do something about it.

DO SOMETHING ABOUT IT

It is time to do something about Git security — and put a stop to bad behavior for good.

There are three options for securing Git:

1. Apply security best practices to native Git.
2. Use a front-end Git tool for additional security features.
3. Switch to a more secure version control tool.

5 Best Practices For Tightening Security in Native Git

There are some security best practices that can be implemented in native Git.

Warning: Implementing these security best practices will require a manual effort. This will take some time, and even then, it's possible to leave a window open for bad behavior.

1. ENSURE SECURE ACCESS

Ensuring secure access is a crucial measure for keeping Git secure. While native Git doesn't provide access control options, access to Git repositories can be controlled in a few ways.

This can be done most simply by setting user permissions, such as:

- Developers: read/write access.
- Users: read-only access.

Another way to ensure secure access is at the server level. This can be done by restricting IP addresses or requiring VPN-only access. It can also be done by having developers work on VDI (virtual desktop infrastructure)

However, these secure access options are light — and not without risk.

While they provide some security measures, they can still allow for insecure access. And with the case of VDI, it makes developers feel like big brother is watching them. It might also be slow, depending on connectivity.

2. DOCUMENT SECURITY POLICIES

Most organizations have documented security policies to protect IP. It is wise for organizations using native Git to document a subset of security policies around Git. This is particularly important for regulated organizations.

Security policies include clear, comprehensive, and well-defined plans, rules, and practices. These policies regulate access to systems and the data in them. It is common to include high availability and disaster recovery policies, along with a plan for the event of a data breach.

Documenting security policies is a good step.

However, it is equally important to make sure employees using Git are aware of the security policy. If a security policy is documented, but no one knows about it, does it matter?

3. AUTHENTICATE USERS PROPERLY

Many Git servers authenticate users by using SSH public keys. These are designed to help prevent "man-in-the-middle" attacks. (These are attacks where a hacker intercepts and possibly alters communications between two parties.)

However, this only works to protect IP when everyone on the team uses SSH keys properly.

That is, if everyone keeps their SSH key private. If they hand the key to someone else — for example, by posting it on a Slack channel to have a colleague help solve a problem — it compromises the system.

That means a hacker could obtain an SSH key and push/pull to/from the repositories.

4. SECURE GIT REPOSITORIES

There are two schools of thought around the best way to secure Git repositories.

Small projects fare better keeping their code in one central Git repository, complete with SSH keys and defined permissions.

Larger projects are often too big for one repository. They often involve multiple development teams working on different pieces. In this case, it is best to break the project up into multiple repositories. Permissions can then be assigned only to the repositories that developers need to access. This approach is often better for regulated organizations with audit concerns.

However, it is far more time-consuming to set up and administer multiple Git repositories manually.

5. USE SSL CERTIFICATES

Using SSL (Secure Sockets Layer) certificates is another way to improve Git security. It typically only takes a few minutes to get an SSL certificate.

There are two options:

- Global SSL certificate.
- Project-level or local SSL certificates.

A global SSL certificate is typically the fastest — and the most prone to carelessness.

A Git server can be put on a virtual machine with everything open (no security groups). This means anyone can access it, even with a SSL certificate in place.

Project-level or local SSL certificates are better, but it will take longer to get a unique SSL certificate for each project.

WHEN SECURITY BEST PRACTICES AREN'T ENOUGH…

Teams turn to adding tools to Git when security practices aren't enough. Many organizations use front-end Git tools to add more layers of security. Some also use network security tools to create defense in depth.

Front-End Git Tools Add Layers of Security

INCLUDING GITHUB, GITLAB, BITBUCKET, AND HELIX TEAMHUB

Front-end Git tools are among the most popular — including GitHub, GitLab, Bitbucket, and Helix TeamHub. Over time, each of these tools have added and enhanced security features.

The security features of front-end Git tools are fairly similar, including:

- Granular roles ensure users can only see the projects they need to see.
- Delegated user management prevents shadow IT.
- LDAP/Active Directory provides more secure authentication.
- 2FA (two-factor authentication) adds another layer of authentication.
- SSO (single sign-on) and SAML (Security Assertion Markup Language) also ensure secure identity and access management.
- IP address whitelisting allows certain users access (while keeping others out).
- Branch protection restricts who can push to branches.

GitHub, GitLab, and Bitbucket allow public repositories. This means that with the flip of a switch, everyone in the world can access a repository. Even read-only access in public repositories could be damaging.

Helix TeamHub, however, does not allow public repositories. Repositories can be shared publicly within a company (where everyone who can see it is identified/logged in) — but kept private from the rest of the world. This is a better option for enterprises concerned about ensuring the security of their repositories and protecting IP.

Using any of these front-end Git tools is a step up from native Git. However, these security features can only do so much, especially if there is only one repository. It is impossible in Git to provide the file/folder level access control that enterprises need. This forces unnecessary splitting of repositories or relaxed security measures.

BUT THESE SECURITY MEASURES OFTEN AREN'T ENOUGH

It is often easy for hackers to break in. And in some cases, they do not need to hack at all. They simply steal credentials.

Infamously, in 2014, data from 50,000 Uber drivers was stolen after credentials were found on public GitHub repositories.

In this case, an Uber employee uploaded the credentials and forget them there. Hackers got access to these repositories, found usernames and passwords, and used them to access internal Uber systems. That is where they found and took PII (Personally Identifiable Information). Uber wound up paying a ransom to these hackers.

The Uber GitHub breach is not an anomaly.

An analysis of public GitHub repositories found that very likely well-meaning employees are unknowingly sharing access codes on GitHub — and making themselves vulnerable to cyberattacks. Developers publish their code to GitHub to share with others and neglect to remove the credentials that are hard-coded into the project.

Typically, this happens more on open source projects. Although, some of these credentials were linked to Fortune 500 companies, payment providers, internet service providers, and healthcare providers.

This is troubling. As companies increase in size, it is more difficult to see what exactly the developers are doing. That is why it is key to amplify security now — beyond the capabilities of front-end Git tools.

How to Truly Lock Down Git and Protect IP

Company IP is important. With Git (both native Git and a front-end Git tool), all a hacker needs is a username and password to pull down an entire repository. In some cases, they might not even need that — which is frightening.

Incorporating multiple layers of security is the best way to protect company IP. By using Helix Core with Helix4Git, teams get multiple layers of security.

HELIX CORE WITH HELIX4GIT DELIVERS UNPARALLELED PROTECTION

Using Helix Core with Helix4Git is the best way to lock down Git and protect IP.

Secure & Integrate With Your IdP

To protect IP, many companies integrate their version control system with Identity Providers (IdPs) to provide 2FA. Other VCS only protect web-based code hosting that does not extend to the command line . This leaves IP exposed — especially on Git servers.

Helix Core is the only version control provider to secure the command line. With [Helix Authentication Service (HAS)](), teams can streamline authentication. It works with clients, plugins, and the command line. HAS offers robust support for OIDC and SAML 2.0 authentication with your IdP of choice, replacing your existing LDAP and/or Active Directory (AD) configuration. This service is internally certified with Microsoft Azure Active Directory (AAD), Okta, and Google Identity. It is also known to be compatible with other IdPs such as Auth0, OneLogin, and Google G-Suite.

Granular Permissions

Access control is an important security feature for version control systems. It is used to define who can access what. This ensures that developers can work on their projects. And it keeps them from seeing projects that they don't need access to.

To truly protect IP, teams need more granular permissions. In native Git, developers typically get access to the entire repository. Even with front-end Git tools, access is typically granted at a repository level.

Helix Core takes permissions a step further.

They can be set at an individual file level, assuring greater security for files. Access can be limited by user and/or IP address. And Helix Core boasts 8 different user access levels — list, read, open, write, review, owner, admin or super.

Immutable Change History

Maintaining a complete change history is another important security feature. It should include who changed what and when. This is especially important when there are multiple or complex changes. Native Git does not protect the change history, making Git particularly dangerous for regulated industries.

Helix Core tracks every change and maintains a complete version history.

This protects the repository. If something goes awry after a change, the repository can be reverted back to its previous state.

In addition to protecting the repository, maintaining an immutable change history is important for passing audits. Regulated industries, in particular, need to maintain a reliable, traceable, and immutable change history for compliance purposes (including PCI DSS and ISO 26262).

Secure Git Repositories

An important security measure for Git repositories is the ability to back them up. However, native Git only backs up on developer laptops.

By using Helix4Git (a Git server inside a Perforce server) with Helix Core, teams gain the backups they need for HA/DR (high availability/disaster recovery).

That means teams using Git can mirror that code into Helix4Git and ensure that it is safe. Developers can gain access to the code. No one sees all of it. And Git code can go into the build pipeline seamlessly, alongside code and other assets from Helix Core.

Get Started With Helix Core With Helix4Git

While many organizations invest in firewalls, IDS (intrusion detection systems), and antivirus software, they neglect to invest in secure version control. This puts their important company IP at risk.

Your IP deserves the best protection. You can get it with Helix Core (and Helix4Git).

Using Helix Core with Helix4Git is the best way to lock down Git. With granular permissions and multiple authentication options, you will be able to ensure that only those who need to access your repositories will be able to. Helix Core also provides a complete history of changes — that no one can ever obliterate.

With all of these security measures together, your valuable IP will receive the best protection possible. To get started, contact us.

CONTACT US

perforce.com/products/helix4git/contact-us

About Perforce

Perforce powers innovation at unrivaled scale. With a portfolio of scalable DevOps solutions, we help modern enterprises overcome complex product development challenges by improving productivity, visibility, and security throughout the product lifecycle. Our portfolio includes solutions for Agile planning & ALM, API management, automated mobile & web testing, embeddable analytics, open source support, repository management, static code analysis, version control, IP lifecycle management, and more. With over 20,000 customers, Perforce is trusted by the world's leading brands to drive their business critical technology development. For more information, visit www.perforce.com.