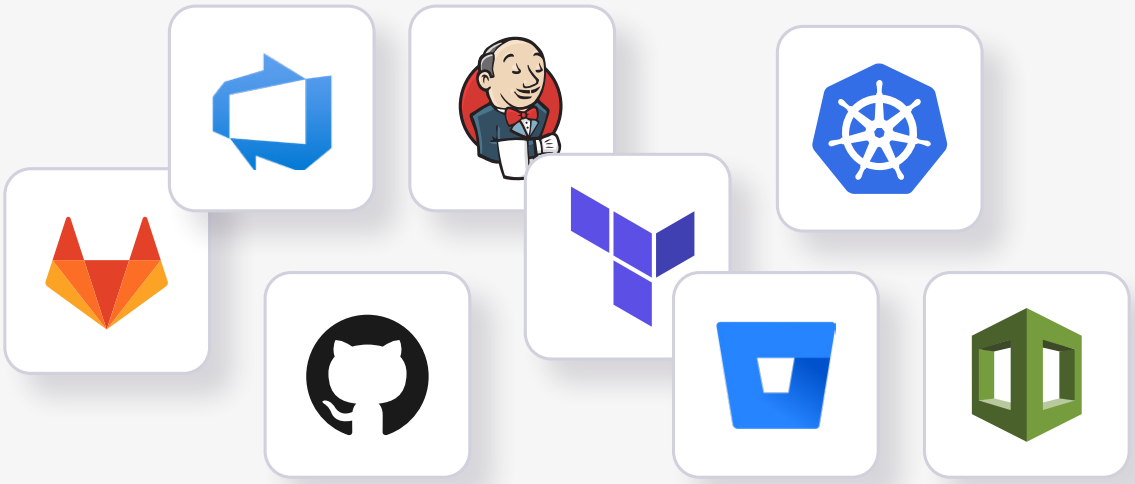
cycode



WHITEPAPER

Secrets Protection:

Top Software Security Best Practices

Table of Contents

Introduction 03

Top Software Security Best Practices

01 — GitLab 04

02 — GitHub 05

03 — Jenkins 06

04 — Terraform 07

05 — Bitbucket 08

06 — Azure DevOps 09

07 — Kubernetes 10

08 — AWS CloudFormation 12

About Cycode 13

Introduction

Top Software Security Best Practices

- GitLab
- GitHub
- Jenkins
- Terraform
- Bitbucket
- Azure DevOps
- Kubernetes
- AWS CloudFormation

About Cycode

Security Best Practices to Safeguard Your Secrets and Source Code

In today's software development lifecycle, your code is being handled by a large number of users and a variety of tools. Dividing the work among a team of developers and relying on software to speed up development processes is a streamlined way to get things done, but provides multiple weak-points for attackers to exploit.

With a plethora of software tools resolving different functions, it can be difficult to make sure each one is running securely. At any moment, a change could introduce a new risk that opens your organization to a malicious attack. Hardcoded secrets, vulnerabilities, misconfigurations, outdated user permissions: These issues not only leave your source code open to suspicious activity, but resolving security concerns eats up valuable resources and development time.

Adopting a security first approach to your SDLC is a great way to get ahead of the misconfigurations and errors, preventing issues before they even arise. But where do you start when you are using so many different tools? Luckily, we are here to help. In this eBook, you will find a collection of the most popular tools and solutions with a list of security best practices to safeguard your secrets and source code.

Introduction

Top Software Security Best Practices

- GitLab
- GitHub
- Jenkins
- Terraform
- Bitbucket
- Azure DevOps
- Kubernetes
- AWS CloudFormation

About Cycode



01 — GitLab

Gitlab is an open-source power player in the SCM market, with the entire company working remotely across 65+ countries. Over the last few years, GitLab's feature set has expanded into numerous adjacent verticals ranging from code security analysis to application deployment automation.

Typically, when feature sets expand, so do problems, so it's always better to review and update security settings.

- Implement secure authentication flow to protect standard credentials
    - Enable 2-Factor Authentication (2FA)
    - Limit access to specific IP addresses by setting "IP Allow List"
    - Enable SAML SSO
- Set up access expiration dates for short-term users and contracts
- Match member permissions to responsibilities through groups or projects
    - Guest-Can see project repository code and can see SDLC related data except code
    - Reporter-Has access to all project data but can't write to the repository
    - Developer-Has read/write access to all the project's data
    - Maintainer-Adds some configuration and management functionality
    - Owner-Full configuration and management functionality
- Use Group Member Lock to block new users from being added to groups and projects
- Disable Forking to prevent code leakage
- Prevent creating public repositories by setting group visibility to private
- Protect main branch by requiring code review before committing code
- Leverage built-in support for security checks
- Periodically review the audit events for anomalous and suspicious activity

Conclusion

As an open-source powerhouse, any vulnerabilities found are going to be exposed for everyone to see. Using open-source solutions quickly speeds up the development process, but it is not without its concerns. That's why securing your GitLab SCM should never be treated as an afterthought.

Introduction

Top Software Security Best Practices

- GitLab
- GitHub
- Jenkins
- Terraform
- Bitbucket
- Azure DevOps
- Kubernetes
- AWS CloudFormation

About Cycode



02 — GitHub

You love it, we love it, everyone loves it. Not surprisingly, most open-source projects live on Github, and it is a streamlined collaboration hub. But, as with any type of massive growth, over time, Github added more features and became more complex. Not to mention, technology and security changed so some of the default settings no longer align with security best practices. You need to review current settings and understand where and how to implement better security at a company-wide level.

- Implement secure authentication flow to protect standard credentials
    - Enable 2-Factor Authentication (2FA)
    - Limit access to specific IP addresses by setting "IP Allow List"
    - Enable SAML SSO (GitHub Enterprise only)
- Set base permissions of every member to “None”
    - Set permissions for each repository based on users or teams
- Tightly control outside collaborators with minimal access and constant monitoring
    - Make sure to inspect access on a per repository basis
- Disable forking
- Disable visibility changes
- Disable public repositories unless your organization requires it
- Enable branch protection rules
- Require code review before merging code to the production branch
    - “Require signed commits” makes attributing reviews to someone easily traceable
- Enable dependency checks on a per-repository basis
- Review the audit log for anomalous or suspicious activities
- Review “Third-party Access” and “Installed GitHub Apps” for unauthorized access

Conclusion

As a very popular open source service, GitHub code is constantly being poked and prodded for security vulnerabilities. Following these best practices ensures your business can continue to leverage open-source code while also protecting yourself from malicious attacks.

Introduction

Top Software Security Best Practices

- GitLab
- GitHub
- **Jenkins**
- Terraform
- Bitbucket
- Azure DevOps
- Kubernetes
- AWS CloudFormation

About Cycode



03 — Jenkins

Jenkins is one of the most well-known tools for creating automation pipelines and integrating them with the rest of your CI/CD tools. It has an active community that has contributed thousands of plugins to extend Jenkins' core functionality, However, Jenkins could also become your organization's weakness without proper controls.

Note – As you will see below, we suggest installing several popular plugins that add to Jenkins's security controls. Even though the recommended plugins are popular with thousands of downloads, it is vital to maintain and update them as they may contain vulnerabilities requiring patching.

- Update vulnerable core and outdated plugins
- Deny malicious access by hardening operating systems
- Use a centralized 3rd party vendor for authentication
- Deploy plugins for more advanced authorization methods
    - Most recognized: Matrix Authorization Strategy, Role-based Authorization Strategy
- Protect the controller node by denying or limiting any code execution
    - No executors on the controller node
    - Disable running builds on the controller node
    - If your environment requires inbound connections, make sure connections are encrypted (use the setting Inbound TCP Agent Protocol/4 (TLS Encryption)
- Disable SSHD
- Set the least privilege for agent permissions for every project
- Minimize the impact of a compromised build by creating a new container each build environment
- Update text rendering to HTML rendering and enable Safe HTML
- Define credentials with Folders plugin
- Use auditing plugins to allow writing or sending logs to a remote server

Conclusion

Jenkins remains a leader for continuous integration software. With the rise of supply-chain attacks in the past years, Jenkins has become an ideal target of choice for threat actors to reach the entire chain of software development, integration, and deployment. Nowadays, Jenkins allows us to put enough security controls to deny this.

Introduction

Top Software Security Best Practices

- GitLab
- GitHub
- Jenkins
- **Terraform**
- Bitbucket
- Azure DevOps
- Kubernetes
- AWS CloudFormation

About Cycode



04 — Terraform

Terraform, developed by Hashicorp, is an infrastructure as code (IaC) framework that allows for declarative resource provisioning. Rather than manipulating configurations manually, Terraform allows for deployment and management of cloud resources with code; this provides the benefits of creating elastically scalable infrastructure, improving consistency in resource management, and enabling software development practices to be employed, including security best practices.

Developing and managing infrastructure is complex, and mistakes can result in the introduction of risk. Employing the best practices described below makes for more secure software and fewer deployment delays due to security concerns. Thus, developers can spend more time building tools and less time remediating errors.

- Execute commands programmatically from a CI/CD tool (Jenkins, CircleCI)
- Perform regular laC misconfiguration scans with pre-apply or post-apply checks
- May be further improved by including critical code monitoring
- Implement a centralized security policy and governance within code
- Enforcing separation of duties helps prevent a compromised account from compromising the whole system
- Adhere to the principle of least privilege
- Utilize version control for the state and store remotely as backup
- .tfstate file should not be committed to source control
- Avoiding storing secrets in state
- Use a backend that supports encryption and control access
- Configure a backend with state locking and remote stage storage
- Modularize when possible using security best practices
- Use variables in Terraform environment to make laC code significantly easier to manage and update

Conclusion

Terraform is a powerful tool for managing your infrastructure. With local or CI/CD scanning of your Terraform code, you get instant feedback on your modules and templates. As your infrastructure grows and Terraform configurations multiply, it's important to secure that infrastructure from development to production.

Introduction

Top Software Security Best Practices

- GitLab
- GitHub
- Jenkins
- Terraform
- **Bitbucket**
- Azure DevOps
- Kubernetes
- AWS CloudFormation

About Cycode



05 — Bitbucket

Collaboration is an essential element when developing any type of new security methodology. Sharing code allows developers to be efficient, but sharing code with your team and any external resources shouldn't mean the entire organization's code can easily be exposed

Platforms like Bitbucket are essential for collaboration, but without proper security measures in place, even the best ways to share can become weapons for hackers. While Atlassian has put some security measures in place to help organizations limit access to code, not everyone knows which to use, or how to implement them. Here's a step-by-step review of Bitbucket security settings to quickly secure your Bitbucket and protect your source code.

- Require all teams, employees and contractors use 2-Step Verification
    - BitBucket Premium: Enforce 2-Step Verification for user's personal account & workspace
- Enable SAML SSO (Atlassian Access users)
- Limit access to specific IP addresses
- Set groups repository access to "None"
    - Make manual access adjustments on a user-by-user basis
    - Review repository access periodically to discover redundant access
- Disable repository forking
- Set visibility level to "False" when creating a new repository
    - Change it prior to releasing repositories to open source
- Disable public repositories across the organization (Bitbucket Server only)
- Enable merge checks (Bitbucket Premium only)
- Encrypt and sign commits or tags with GPG keys (Bitbucket Server only)
- Review the audit log for anomalous or suspicious activities
- Inspect installed third-party apps in your workspace
    - Review both the "Installed apps" and "Enable development mode" to ensure no unauthorized access is granted

Conclusion

In general, it's more efficient to be able to continuously monitor and inspect the security posture across various Bitbucket security settings. In a pinch, these best practices are a great foot forward in securing your favorite collaboration platform.

Introduction

Top Software Security Best Practices

- GitLab
- GitHub
- Jenkins
- Terraform
- Bitbucket
- **Azure DevOps**
- Kubernetes
- AWS CloudFormation

About Cycode



06 — Azure DevOps

Microsoft Azure has been a popular platform for various kinds of cloud computing for years. The service caters to developers and their DevOps teams, weaving tightly into an organization's software supply chain and lifecycle.

Here are best practices for securing your code within the Azure DevOps platform to ensure you get the most out of the service without putting your code at risk.

- Connect Active Directory organization to support advanced login restrictions
- By default, the web platform leverages Microsoft accounts for authentication
- Set up Multi-Factor Authentication (MA)
- Furthemore, set policies and limit access by location
- Leverage SO configuration and enable Security Assertion Markup Language
- (SAML) for Single Sign-On (SSO)
- Determine which users will be granted Basic or Stakeholder permissions
- Divide users and groups of users based on permission levels
- General, Boards, Analytics, and Test Plans
- Configure the repository's granular project permissions
- Needs will vary on a project-to-project basis
- Disable Forking to keep better track of repository security
- Adjust visibility at top level of project settings
- Protect branches by prohibiting any code changes without review
- Enable "Require a minimum number of reviewers" setting
- Also recommend enabling "Commit author email validation" and "File path validation"
- Review audit events for anomalous or suspicious activities

Conclusion

As with any cloud-based solution, security must always be the topmost concern. This is especially true for Azure DevOps, but with proper practices, you'll be able to keep your code locked tight and your lifecycle free of malicious activities.

Introduction

Top Software Security Best Practices

- GitLab
- GitHub
- Jenkins
- Terraform
- Bitbucket
- Azure DevOps
- **Kubernetes**
- AWS CloudFormation

About Cycode



07 — Kubernetes

Kubernetes is a powerful tool allowing for orchestration of containerized services, applications, and workloads. While Kubernetes has become widely deployed in the last several years, security has lagged behind and best practices are only beginning to coalesce now. According to a 2021 Red Hat survey, 55% of respondents have had to delay deploying a Kubernetes application into production due to security concerns; nearly a third of such security concerns even occurred during runtime.

Even more notable is that 94% of respondents experienced one or more security incidents within their Kubernetes environments–nearly every respondent to the survey had recently experienced a Kubernetes security incident. This is clear evidence that more guidance is needed to help prevent such security vulnerabilities.

- Use version control for configuration files related to deployment and services
- This provides a log of who made changes and possible misconfigurations
- Isolate Kubernetes code from feature code in dedicated repositories
- Consider utilizing a package manager, such as Helm
- Conduct readiness and liveness probes as part of regular check-ups
- Readiness probe – used to determine if the pod is initialized
- Liveness probe – used to check whether pods are serving requests properly
- Audit logs regularly as part of efforts to identify potential vulnerabilities
- Minimize included parts to reduce security risks and increase build and pull speed
- Alpine images are recommended as they are 10x smaller than the base image
- Utilize namespaces to organize your Kubernetes cluster
- Enable role-based access control
- Ensure the production cluster is running on the latest stable version

A Note on IaC Code - There is room for organization choice regarding placement of Kubernetes declaration files and docker file code. Placing IaC code in a separate repository allows for least privilege policies to be enforced as means of reducing the exposure risk in case of a breach; this allows for separation of duties in a way that meets compliance needs.

This also allows for code and configurations to be independently rolled back and for easier matching of changes to a running environment. However,

Introduction

Top Software Security Best Practices

- GitLab
- GitHub
- Jenkins
- Terraform
- Bitbucket
- Azure DevOps
- **Kubernetes**
- AWS CloudFormation

About Cycode



07 — Kubernetes (cont.)

placing IaC with feature code allows for better reproduction of production environments for bug fixing–CI/CD tools are usually set up to work with infrastructure-as-code definitions contained in the same repo as feature code. This grouping also allows for easier rollbacks to previous versions, providing benefit from the perspective of version control simplicity.

Ultimately, this choice needs to be made early in the organization's life and consistency should be enforced to reduce the chances of poor practices being employed in the development life cycle.

Conclusion

Kubernetes provides many options to assist deployment. Adhering to these best practices for Kubernetes improves the uptime of services, begets better reliability, and makes for a more solid foundation. However, it is recommended to also implement security processes into the continuous integration pipeline as well as taking steps to automate the process.

Introduction

Top Software Security Best Practices

- GitLab
- GitHub
- Jenkins
- Terraform
- Bitbucket
- Azure DevOps
- Kubernetes
- **AWS CloudFormation**

About Cycode



08 — AWS CloudFormation

AWS CloudFormation gives organizations the ability to easily manage a collection of AWS resources by automating the initialization, provisioning, and deletion of infrastructure, services, and applications. CloudFormation may be broken down into two parts: templates and stacks. A template is a file that defines which resources are required to run your application. With such a powerful tool at hand, developers need to be aware of security best practices to help mitigate the cyber security risk.
In order to secure this tool, security best practices for AWS CloudFormation should be adhered to as misconfigurations are amplified within IaC environments. We recommend the following to help mitigate risk:

- Ensure consistent governance through AWS CloudFormation Stack Policies
  - Specify a stack policy when creating stacks containing critical resources
  - Utilizing DeletionPolicycan help preserve resources when its stack is deleted
- Control access with IAM
  - Helps adhere to principles of least privilege
- Avoid hard coding secrets in AWS CloudFormation stack template
- Enable stack notifications to monitor stack events
  - Use a Lambda function as an intermediary between CloudFormation and SNS
- Secure CloudFormation by detecting and correcting environmental drift
  - Utilize the drift detection feature to detect unmanaged configuration changes
- Log API calls with AWS CloudTrail
  - Caution: Configure the S3 bucket securely as insecure configurations can result in a code leak of the logs
- Regularly scan for code leaks
- Block public access settings page for AWS S3 buckets
- Update Amazon EC2 Linux instances regularly

Conclusion

AWS CloudFormation is a powerful tool for managing infrastructure. As an organization's infrastructure grows and the quantity of CloudFormation stacks and templates increases, it's important to secure that infrastructure from development to production. In order to get a holistic view into what is actually being provisioned or changed, including variables being called, you may need to scan the AWS CloudFormation resource outputs.

Introduction

Top Software Security Best Practices

- GitLab
- GitHub
- Jenkins
- Terraform
- Bitbucket
- Azure DevOps
- Kubernetes
- AWS CloudFormation

About Cycode

About Cycode

One thing all these best practices have in common is that they require manual action, which is cumbersome and does not scale well. In general, it's more efficient to be able to continuously monitor and inspect your security posture across various settings and tools.That's where a comprehensive security platform like Cycode can help.

Cycode provides complete visibility into enterprise DevOps tools and infrastructure, ensuring that strong authentication and secure development practices are in place. We analyze actual usage and recommend tightening the access model to fit a least privileged approach. We monitor the SCM activity in real time, allowing us to alert you on various security-related incidents based on the policy that best fits your organization.

Cycode's capabilities include comprehensive security scanning to prevent exposed secrets, environmental drift, code leaks, and other common issues, all within developer-friendly workflows. Once integrated into developer workflows, each commit and PR is scanned for issues including hardcoded secrets, known vulnerabilities, or potential misconfiguration. Alerts are issued appropriately. Remediation advice is available both through a GUI and within the PR.

Bottom line:
The best practice for securing your SDLC is to use Cycode to protect your organization's source code wherever it is sprawled.

Learn More →

More Resources you might find useful:



Hidden Dangers of **Hardcoded Secrets in AI Generated Code**

Watch Now →



Hardcoded **Secrets Detection**

Learn More →



A Risk-Based Approach to **Eliminating Hardcoded Secrets**

Read Now →

cycode

Need more help? Feel free to Contact Us!