Best practices for securing your build system

In this article

Guidance on how to protect the end of your supply chain—the systems you use to build and distribute artifacts.

About this guide

This guide describes the highest impact changes you can make to improve the security of your build systems. Each section outlines a change you can make to your processes to improve security. The highest impact changes are listed first.

What's the risk?

Some attacks on software supply chains target the build system directly. If an attacker can modify the build process, they can exploit your system without the effort of compromising personal accounts or code. It's important to make sure that you don't forget to protect the build system as well as personal accounts and code.

Secure your build system

There are several security capabilities a build system should have:

1. The build steps should be clear and repeatable.
2. You should know exactly what was running during the build process.
3. Each build should start in a fresh environment, so a compromised build doesn't persist to affect future builds.

GitHub Actions can help you meet these capabilities. Build instructions are stored in your repository, alongside your code. You choose what environment your build runs on, including Windows, Mac, Linux, or runners you host yourself. Each build starts with a fresh runner image, making it difficult for an attack to persist in your build environment.

In addition to the security benefits, GitHub Actions lets you trigger builds manually, periodically, or on git events in your repository for frequent and fast builds.

GitHub Actions is a big topic, but a good place to get started is "Understanding GitHub Actions," as well as "Workflow syntax for GitHub Actions," and "Triggering a workflow."

Sign your builds

After your build process is secure, you want to prevent someone from tampering with the end result of your build process. A great way to do this is to sign your builds. When distributing software publicly, this is often done with a public/private cryptographic key pair. You use the private key to sign the build, and you publish your public key so users of your software can verify the signature on the build before they use it. If the bytes of the build are modified, the signature will not verify.

How exactly you sign your build will depend on what sort of code you're writing, and who your users are. Often it's difficult to know how to securely store the private key. One basic option here is to use GitHub Actions encrypted secrets, although you'll need to be careful to limit who has access to those GitHub Actions workflows. If your private key is stored in another system accessible over the public internet (like Microsoft Azure, or HashiCorp Vault), a more advanced option is to authenticate with OpenID Connect, so you don't have to share secrets across systems. If your private key is only accessible from a private network, another option is to use self-hosted runners for GitHub Actions.

For more information, see "Using secrets in GitHub Actions", "About security hardening with OpenID Connect", and "About self-hosted runners."

Harden security for GitHub Actions

There are many further steps you can take to additionally secure GitHub Actions. In particular, be careful when evaluating third-party workflows, and consider using `CODEOWNERS` to limit who can make changes to your workflows.

For more information, see "Security hardening for GitHub Actions;" particularly "Security hardening for GitHub Actions" and "Security hardening for GitHub Actions."

Next steps

- "Securing your end-to-end supply chain"
- "Best practices for securing accounts"
- "Best practices for securing code in your supply chain"