Hardening Your TeamCity Server

TeamCity is at the heart of your build process. It builds your source code into deployable artifacts and often also deploys those artifacts, which means it has potential access to sensitive information.

While it offers great security by default, here are some additional steps you can take to harden the security of your build pipelines.

General Advice

Update your TeamCity server regularly

We strongly recommend that you regularly update TeamCity to the latest released version.

TeamCity will automatically notify you via the UI once a new update is available. You can also manually check for new TeamCity versions under **Server Administration > Updates** for TeamCity itself and under **Server Administration > Plugins** for any available plugin updates.

From a **technical perspective**, upgrades between bugfix releases within the same major/minor version are backwards compatible (e.g. 2020.1.1 → 2020.1.2) and support relatively simple rollbacks. For all other major upgrades, we do our best to ensure that they run as smoothly as possible, though backups are strongly recommended for easy rollbacks.

From a **licensing perspective**, upgrades between bugfix releases are also safe. If your license covers 2020.1.1, then you will be able to upgrade to *any* 2020.1.x version.

Subscribe to the security notification service

We also recommend that you subscribe to the security notification service to obtain the latest information about security issues that may affect TeamCity or any other JetBrains products.

Credentials

Use strong credentials, and use them carefully

We recommend using strong credentials not only for your TeamCity server, but also for all other services that are involved in a build or that your software requires in production.

Make especially sure to keep your credentials out of:
- Repositories, such as GitHub, GitLab, etc.
- Environment variables, as they're often logged or shared with third-party monitoring systems.
- The Build log – make sure you don't randomly log sensitive information.

Also, if you're using Versioned Settings (in Kotlin DSL or XML format), never store your credentials in your configuration files. Instead, use tokens.

Store secure data with the password parameter type

To store passwords or other secure data in TeamCity settings, you are strongly advised to use TeamCity's password parameter type. This will make sure that sensitive values never appear in TeamCity's Web UI and that they will also be asterisked in the build log.

Use a secrets management tool

Although password parameters are masked in the UI, encrypted at rest, and protected from being exposed in the build log as plain text, this often does not provide a high enough level of security.

You may consider using a tool such as HashiCorp Vault, which lets you manage and rotate all the sensitive credentials you'll be using in a build and which integrates well with TeamCity.

Use external authentication

If applicable, use one of our external authentication modules, ranging from LDAP and Windows Domain integration to authenticating via GitHub, GitLab, or others. You can then disable the built-in authentication of TeamCity, so that TeamCity will no longer keep hashed passwords in the internal database.

Use a custom encryption key

Passwords that are necessary to authenticate in external systems (like VCS, issue trackers, and so on) are stored in a scrambled form in <TeamCity Data Directory> and can also be stored in the database. However, the values are only scrambled, which means they can be retrieved by a user who has access to the server file system or database.

Instead of this default scrambling strategy, you can consider enabling a custom encryption key. In this case TeamCity will use your unique custom key to encrypt all secure values, instead of using the default scrambling mechanism.

Permissions

Use predefined roles

Out of the box, TeamCity offers several predefined roles:
- System Admin
- Project Admin
- Project Developer
- Project Viewer

Create user groups that match your organizational structure and assign the above roles to those groups. Then add your users to the respective groups, granting them the lowest level of privileges they need for their day-to-day work.

It is also strongly recommended that you create new roles with additional permissions, instead of immediately assigning the project-admin role to anyone who needs slightly more privileges. (This does not work if you disable per-project permissions.)

Use per-project authorization

To tighten security even more, you can also make use of per-project authorization. This way, your developers could, for example, have access only to the compilation part of your build chain, while devops could access and run the deployment part.

Do not enable Guest Login

By default, logging into TeamCity anonymously is disabled. Make sure not to enable it on production TeamCity server instances that are exposed to the internet, unless you want external users to be able to see all your builds and the associated log files/artifacts.

Create a separate REST user

If you access TeamCity's REST API from an external script or program, we recommend you create a separate user with a limited number of permissions for it. It would also be wise to create auto-expiring access tokens, instead of using the user's username/passwords to access the API.

Restrict deployment build permissions

Make sure that your deployment build chains do not allow personal builds. Limit the number of developers who can trigger those builds, and use a separate pool of clean agents for those builds.

TeamCity Server

Protect TeamCity's data directory

Users who have read access to <TeamCity Data Directory> can access all the settings on the server, including configured passwords. Hence you need to make sure to make this directory only readable by OS users who are actually administrators of the services.

Protect your TeamCity server

In general, limit access to the machine your TeamCity server runs on. Enable access logs and regularly review them.

Use HTTPS everywhere

It is recommended that you enable HTTPS for TeamCity. We currently recommend enabling HTTPS on your reverse proxy (like Nginx or Apache).

Secure your external database

Make sure to use a dedicated database user account with strong credentials for your TeamCity server's database schema. Consider using database encryption if your database supports it.

Version Control

Use a recent version of Git

Make sure to always use the latest stable Operating System and Git version on your build agents. Update regularly.

Properly manage your SSH keys

If you are using SSH keys to access your repositories, do not store them on your build agents. Instead, use TeamCity's SSH Keys Management facilities and upload them to the TeamCity server.

Also, instead of disabling known hosts checks, make sure to maintain an .ssh/known_hosts file on the server and build agents for every host you are connecting to.

Use a dedicated VCS user

If you are not using advanced features like the Kotlin DSL or, in general, if you don't need to commit to your repository as part of your build process, we recommend keeping a dedicated VCS user without write permissions to connect to your repositories.

Build Agents

Run clean production builds

We recommend enabling the Enforcing Clean Checkout option for your production builds, so as to prevent tampering with source code on an agent.

Use disposable, network-protected build agents

If possible, try using disposable, one-off build agents. The shorter the agent's lifetime, the smaller the chance of compromise. Also make sure to use OS-dependent firewall rules to disable incoming network access for your cloud agents.

Use agent pools for different projects

If you run several agents on the same machine and do not have the Enable Clean Checkout option set, beware that compromised agents or untrusted projects could potentially modify source code in "neighbor" working directories.

To mitigate this risk, consider running just one agent per machine and use different agent pools for different (private/public) projects.

Integrations

Don't blindly build public pull requests

If you build pull requests from unknown users or users outside of your organization, be aware that pull requests could contain malicious code that would be run on your build agent.

Either disallow building public pull requests, or use separated, isolated, throw-away agents. TeamCity also offers a built-in health report, which detects and reports pull request builds.

Be aware of the security implications when using Versioned Settings

When you use Versioned Settings (Kotlin DSL, XML) and those settings are placed in the same repository as the source code, any malicious developer can potentially modify and leak project configuration settings. This could be done, for example, by adding a build step that prints out passwords or sends them somewhere as a file.

As an option, you could use a separate repository that only a limited number of users can commit to for your versioned settings.

Be careful with third-party plugins

When installing plugins, make sure they come from a trusted source and that their source code is available. Plugins can potentially access all information on a TeamCity server, including sensitive information.

Artifact Storage

Disable anonymous access

Regardless of where you store your build artifacts (such as S3), make sure to disable anonymous access to your storage location.

Use proper access policies

Use proper access policies to protect your S3 or other storage locations / repositories for artifacts. Also use encryption if possible. Check, monitor, and regularly review the access logs of your storage locations.

Don’t put sensitive data into artifacts

This goes without saying, but do not store credentials or other sensitive information as plain text in your build’s artifacts.

Build History & Logs

Keep your build history

Keep your build history and logs for a longer period of time, especially for builds doing critical deployments, by specifying corresponding Clean-Up rules for your project. Also, make sure to not grant the “remove build” permissions to developers, as this would circumvent the archiving.

Both measures may help with tracing malicious activities, even if they happened a long time ago.

Archive server and agent logs

Collect TeamCity server and build agent logs in an archive and put them under properly secured storage.