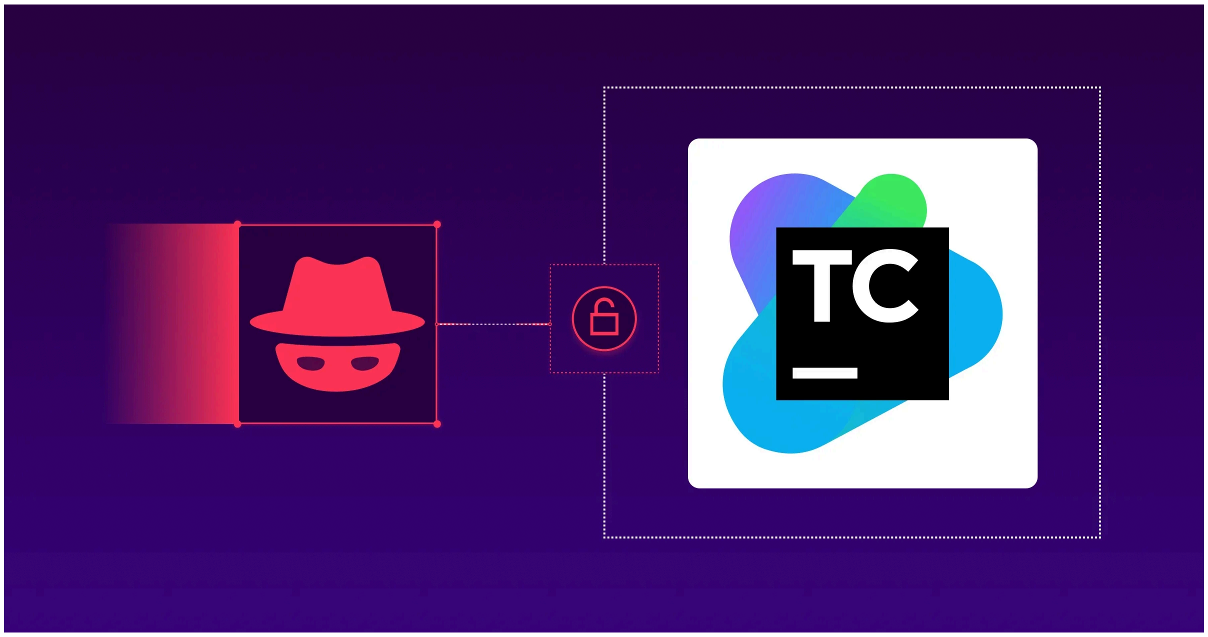
Blog post

Source Code at Risk: Critical Code Vulnerability in CI/CD Platform TeamCity

Stefan Schiller
VULNERABILITY RESEARCHER

September 26, 2023
7 MIN READ



Update 2023-09-27: Full technical details added (see *Technical Details* section).

Key Information

- Sonar's Vulnerability Research Team has discovered a critical security vulnerability in TeamCity, a popular Continuous Integration and Continuous Deployment (CI/CD) server from JetBrains.
- The discovered vulnerability tracked as CVE-2023-42793 allows unauthenticated attackers to execute arbitrary code on the TeamCity server (remote code execution, RCE).
- Attackers could leverage this access to steal source code, service secrets, and private keys, take control over attached build agents, and poison build artifacts.
- JetBrains released a dedicated blog post providing comprehensive information about the vulnerability.
- The vulnerability was fixed with TeamCity version 2023.05.4.

Introduction

TeamCity is a widely used Continuous Integration and Continuous Deployment (CI/CD) server from JetBrains deployed by more than 30,000 customers worldwide. The application can either be used via the cloud-hosted solution TeamCity Cloud or deployed on an own server via TeamCity on-premises. According to Shodan, more than 3,000 of these on-premises servers are directly exposed to the Internet.

CI/CD servers like TeamCity are used to automate the process of building, testing, and deploying software applications. This means that these servers have access to one of the most valuable assets of a company: source code. Since they are also responsible for building and deploying this source code, they not only store sensitive secrets and keys but also control the build artifacts, which become part of a software release. This makes CI/CD servers a high-value target for attackers.

In this article, we explain the code vulnerability we discovered in TeamCity, determine the root cause of it, and describe how this and similar vulnerabilities can be prevented.

Impact

TeamCity server version **2023.05.3 and below** is prone to an authentication bypass, which allows an **unauthenticated attacker** to gain **remote code execution (RCE)** on the server. This enables attackers not only to steal source code but also stored service secrets and private keys. And it's even worse: With access to the build process, attackers can inject malicious code, compromising the integrity of software releases and impacting all downstream users. The attack does **not** require any user interaction:

We want to emphasize the importance of prompt action to mitigate this risk. Because this vulnerability does not require a valid account on the target instance and is trivial to exploit, it is likely that this vulnerability will be exploited in the wild.

We strongly advise all TeamCity users to apply the latest patch provided by JetBrains as soon as possible. The first release known to address the vulnerability is TeamCity version 2023.05.4. TeamCity Cloud is not affected by the vulnerability.

Indicators of Compromise

The existence of an authentication token named `RPC2` is a strong indicator of compromise. A token with this name was very likely created by an unauthorized and potentially malicious user to gain access to the server:

Please notice that an attacker may have deleted or renamed the token after gaining a foothold on the server.

Technical Details

In the interest of responsible disclosure and ethical reporting, it’s crucial to emphasize that the technical details of this critical vulnerability were disclosed only after careful consideration and the public release of a corresponding exploit. Every effort was made to ensure that JetBrains had adequate time and information to address and remediate the vulnerability. The goal is not only to highlight the potential risks and solutions but also to collaborate towards a safer and more secure digital landscape for all stakeholders involved.

Request Interceptors

TeamCity uses request interceptors in order to perform specific actions for **every HTTP request**. One of these actions implemented via a request interceptor is the authorization mechanism.

The class responsible for applying this and other interceptors is called `RequestInterceptors`. When a request is received, the `preHandle` method of this class is invoked, which determines if the request is suitable for pre-handling by calling `requestPreHandlingAllowed`:

jetbrains.buildServer.controllers.interceptors.RequestInterceptors

```
public final boolean preHandle(HttpServletRequest req, ...) {
    if (!this.requestPreHandlingAllowed(req)) {
        return true;
    }
    // ...
}
```

Amongst other things, this method checks if the requested path matches a predefined list of path expressions (`myPreHandlingDisabled`). For matching paths, no pre-handling should be applied:

jetbrains.buildServer.controllers.interceptors.RequestInterceptors

```
private boolean requestPreHandlingAllowed(@NotNull HttpServletRequest req) {
    // ...
    if
(!this.myPreHandlingDisabled.matches(WebUtil.getPathWithoutContext(req))) {
        return true;
    }
    // path matches myPreHandlingDisabled? no pre-handling!
    return false;
}
```

In the constructor of `RequestInterceptors` , two path expressions are added, which should be excluded from any pre-handling processing:

jetbrains.buildServer.controllers.interceptors.RequestInterceptors

```
public RequestInterceptors(@NotNull List<HandlerInterceptor> var1) {
    // ...
    this.myPreHandlingDisabled.addPath("/**" +
XmlRpcController.getPathSuffix());
    this.myPreHandlingDisabled.addPath("/app/agents/**");
}
```

The first path expression starts with the static string `"/**"` , followed by the return value of `XmlRpcController.getPathSuffix()` , which returns the static string `"/RPC2"` :

jetbrains.buildServer.controllers.XmlRpcController

```
public class XmlRpcController extends AbstractController {
    public static String getPathSuffix() {
        return "/RPC2";
    }
    // ...
}
```

Thus, the resulting path expression is `"/**/RPC2"` . For requests to a path matching this expression, no pre-handling interceptors are applied. **This also means that for these requests, no authorization check is performed**.

This is particularly dangerous because this expression allows arbitrary prefixes in the requested path due to the two asterisks (`"/**/"`). This effectively disables the authorization check for every request to a path ending with `/RPC2` .

Request Path Parameters

TeamCity provides a REST API for integrating external applications. The available endpoints are documented here. One of these endpoints allows the creation of a user authentication token via the route `/app/rest/users/<userLocator>/tokens` . Since this endpoint route ends with the static suffix `"/tokens"` , it cannot be used to bypass the authentication.

However, the documentation does not contain all endpoints. There are additional hidden endpoints. One of these is a slightly different version of the token creation endpoint:

jetbrains.buildServer.server.rest.request.UserRequest

```
@Api("User")
@Path(UserRequest.API_USERS_URL)
public class UserRequest {
    // ...
    @Path("/{userLocator}/tokens/{name}")
    @ApiOperation(value = "Create a new authentication token for the
matching user.", nickname = "addUserToken", hidden = true)
    @POST
    @Produces({"application/xml", "application/json"})
    public Token createToken(@PathParam("userLocator") @ApiParam(format =
"UserLocator") String userLocator, @PathParam("name") @NotNull String name,
...) {
        // ...
        SUser user = this.myUserFinder.getItem(userLocator, true);
        AuthenticationToken token =
tokenAuthenticationModel.createToken(user.getId(), name, ...);
        return new Token(token, ...);
    }
}
```

This endpoint also creates a user authentication token, but it additionally allows the provision of a name for this token via the `{name}` request path parameter. Since this name can be arbitrarily set, `RPC2` is considered valid:

Thus, an unauthenticated attacker can create a new authentication token for any user via the following request:

```
POST /app/rest/users/<userLocator>/tokens/RPC2
```

The response to this request contains the authentication token for the user specified via the `<userLocator>` (e.g., `id:1` for the default admin account). This token can then be used to access the application.

While we won't be sharing exploitation details, with access to the admin account, there are various ways to execute arbitrary code on the server.

Learnings

Authorization checks are usually applied to endpoint handlers individually. This might be as simple as adding a specific decorator or deriving the controller class from a predefined authenticated-only base controller class. TeamCity took an even more secure approach: all endpoints require the user to be authenticated by default. If an endpoint should be made available without authentication, this needs to be explicitly defined in the endpoint handler.

This secure-by-default approach is the preferred way, but it still has a blind spot: global request interceptors. Depending on the programming language and framework, these are usually called middleware, filters, hooks, or interceptors. The purpose of them is to perform specific actions for every HTTP request. Because they are implemented in a separate class or function independent of the specific endpoint handlers, they are often overlooked during security assessments. Whether you are looking at it from the defensive or offensive perspective: always consider these global request interceptors as part of the exposed attack surface!

Another sensitive aspect from a security point of view is the usage of wildcard expressions. These are used in scenarios where a static value is not sufficient to represent all acceptable inputs. The downside of this is that an expression chosen too unrestrictively allows more than actually intended. In this case, the `"/**/RPC2"` wildcard was never supposed to also include the REST API endpoints. To prevent these kinds of issues a generally good approach is to be as restrictive as possible.

Patch

The vulnerability was fixed with TeamCity version 2023.05.4. By now, the only way the `/RPC2` endpoint should be accessed is directly without any prefixes in the requested path. The patch removes the wildcard expression for the `/RPC2` pre-handling exception:

jetbrains.buildServer.controllers.interceptors.RequestInterceptors

```
public RequestInterceptors(@NotNull List<HandlerInterceptor> var1) {
    // ...
-   this.myPreHandlingDisabled.addPath("/**" +
XmlRpcController.getPathSuffix());
+   this.myPreHandlingDisabled.addPath(XmlRpcController.getPathSuffix());
}
```

This way, pre-handling is only disabled when directly accessing `/RPC2` without any additional prefixes in the requested path and cannot be leveraged to bypass the authentication for other endpoints.

Timeline

Our Vulnerability Research team stood in close communication with JetBrains, and we would like to thank them for their efficient collaboration:

DATE	ACTION

2023-09-06, 10:44 CET	We report the issue to JetBrains.
2023-09-06, 12:39 CET	JetBrains confirms receipt of the report.
2023-09-06, 12:54 CET	JetBrains reproduces the issue.
2023-09-07	JetBrains fixes the issue in 2023.05 branch.
2023-09-12	JetBrains prepares the plugin that could be used as a workaround.
2023-09-14	JetBrains sends an update: The issue has been reproduced and confirmed to be a major security issue.
2023-09-18	TeamCity version 2023.05.4 is released, which fixes the vulnerability.
2023-09-18	JetBrains sends notifications to customers asking them to update as soon as possible.
2023-09-19	CVE-2023-42793 is published.
2023-09-21	Coordinated release of first blog posts from JetBrains and Sonar.

2023-09-27	Full disclosure after a public exploit was released.

Summary

In this article, we outlined the impact of a critical vulnerability we discovered in the popular CI/CD server TeamCity. We determined the root cause of the vulnerability and outlined how attackers could leverage it. Furthermore, we provided general recommendations on preventing these kinds of issues and looked at the patch applied to fix the vulnerability.

At last, we would like to give a huge shoutout to JetBrains, who quickly confirmed the vulnerability, informed all affected users, and provided a fix. Thank you!

Related Blog Posts

- Agent 007: Pre-Auth Takeover of Build Pipelines in GoCD
- Agent 008: Chaining Vulnerabilities to Compromise GoCD
- Securing Developer Tools: A New Supply Chain Attack on PHP
- Securing Developer Tools: OneDev Remote Code Execution
- Securing Developer Tools: Argument Injection in Visual Studio Code

SHARE

Sonar Solutions

SAST
What is clean code
Power of clean code
Clean as you code
AI-assisted & quality-assured code
DevOps transformation
Outsourcing software development
Reduce & manage technical debt
Secure by design
Code coverage
Code review
For developers
For enterprise
Infrastructure as code
Public sector

Products

SonarQube for IDE
SonarQube Server
SonarQube Cloud

Pricing

Start for free
Explore pricing

Company

About
Careers
Commitment to open source
Customers
Partners
Contact us
Accessibility
NEW! Brand identity

Media

Coverage
Press releases

Resources

Events hub
Customer stories
White papers
Learn
Community
Support

Knowledge

Explore Sonarpedia
Blog
Languages
SonarQube Server Documentation
SonarQube Cloud Documentation
SonarQube for IDE Documentation