Penetration Test on Backstage for the Backstage team

Final Report and Management Summary

2022-08-24

X41 D-SEC GmbH
Krefelder Str. 123
D-52070 Aachen
Amtsgericht Aachen: HRB19989

https://x41-dsec.de/
info@x41-dsec.de

Organized by the Open Source Technology Improvement Fund

Revision	*Date*	*Change*	*Author(s)*
1	2022-05-25	Preliminary Report	M. Vervier
2	2022-08-18	Public Report	E. Sesterhenn, Y. Klawohn, M. Vervier

Contents

Dashboard

Target	
Customer	the Backstage team
Name	Backstage
Type	Web Application
Version	v0.70.0 (04bb0dd82) and 1.4.0 (2231987)
Engagement	
Type	White Box Penetration Test
Consultants	4: Luc Gommans, Dr. Alexander Pirker, Yasar Klawohn, Eric Sesterhenn, and Markus Vervier
Engagement Effort	38 person-days, 2022-03-03 to 2022-08-11
Total issues found	12

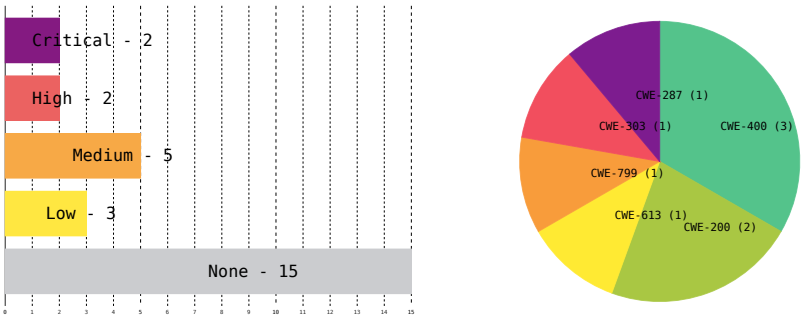


Figure 1: Issue Overview (l: Severity, r: CWE Distribution)

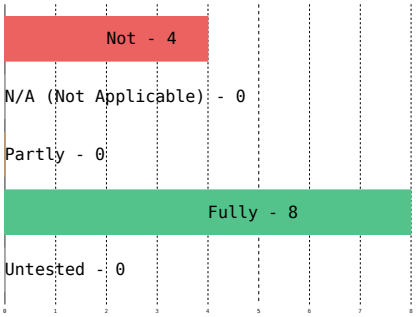


Figure 2: Remediation Status

1 Executive Summary

In March and April 2022, X41 D-Sec GmbH performed a White Box Penetration Test against Backstage to identify vulnerabilities and weaknesses in the platform version 0.70.0. In August 2022, X41 inspected the fixes for the discovered issues applied until version 1.4.0.

The test was organized by the Open Source Technology Improvement Fund[1], a corporate non-profit dedicated to securing open source applications and infrastructure.

A total of twelve vulnerabilities were discovered during the test by X41. Two were rated as critical, two were classified as high severity, five as medium, and three as low. Additionally, 15 issues without a direct security impact were identified.

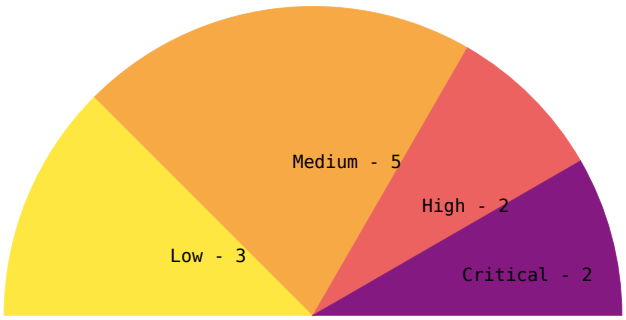


Figure 1.1: Issues and Severity

Backstage is a software catalog and development platform that enables teams to quickly ship high-quality code.

[1] https://ostif.org

In a White Box Penetration Test, the testers receive all available information about the target, including source code. The test and retests were performed by three experienced security experts between 2022-03-03 and 2022-08-11.

The most severe issues are related to the support of multiple authentication providers. These allow the login of users that are not registered at Backstage but only with the third party providers, leading to potential authentication and authorization problems. Additionally, a misconfiguration of the JWT token used for session identification was identified, which allows the impersonation of other users with a valid JWT token.

After the initial test, the issues were reported and discussed with the developers. Subsequently, fixes and mitigations were developed by the Backstage team and verified by X41.

X41 recommends to include the reasoning for issues risk accepted by the Backstage team into the end user documentation as recommendations or warnings. This will ensure that users of Backstage can make informed decisions to ensure a safe operation.

In conclusion the code of Backstage is considered to be of good quality given the complexity of the system and its purpose. The interaction with the Backstage team was smooth and all issues could be discussed openly.

2 Introduction

X41 reviewed Backstage, setup documentation, and a number of its plugins. The software is used to build developer portals to create an overview of the different software running in an organization, integrating with external systems for features like Continuous Integration.

Compromising Backstage in an organization would allow an attacker to capture user authentication tokens — as they use OAuth — or other mechanisms to trigger actions on those systems.

2.1 Methodology

X41 reviewed Backstage as per the scope defined below. The review was mainly based on a source code review alongside with analyses of local installations of the software on test systems. Default installations, where the setup instructions are followed correctly, were checked for weaknesses such as unnecessarily broad permissions or missing API[1] restrictions.

A manual approach for penetration testing and for code review is used by X41. This process is supported by tools such as static code analyzers and industry standard web application security tools[2].

X41 adheres to established standards for source code reviewing and penetration testing. These are in particular the *CERT Secure Coding*[3] standards and the *Study - A Penetration Testing Model*[4] of the German Federal Office for Information Security.

[1] Application Programming Interface
[2] `https://portswigger.net/burp`
[3] `https://wiki.sei.cmu.edu/confluence/display/seccode/SEI+CERT+Coding+Standards`
[4] `https://www.bsi.bund.de/SharedDocs/Downloads/EN/BSI/Publications/Studies/Penetration/penetration_pdf.pdf?__blob=publicationFile&v=1`

2.2 Findings Overview

DESCRIPTION	SEVERITY	REMEDIATION	ID	REF
No Expiration Claim for JWT Access Token	MEDIUM	FULLY	SPBS-CR-22-01	4.1.1
Constant Subject Claim for JWT Access Token	HIGH	NOT	SPBS-CR-22-02	4.1.2
Missing Rate Limiting for Backend Components	LOW	NOT	SPBS-CR-22-03	4.1.3
Unauthenticated DoS Situation Due to Scaffolder Event Streaming	MEDIUM	NOT	SPBS-CR-22-04	4.1.4
Information Leakage through Misconfiguration via Templates	LOW	FULLY	SPBS-CR-22-05	4.1.5
Potential Information Leakage through Symlinks in Repositories	HIGH	FULLY	SPBS-CR-22-06	4.1.6
DoS Situation for catalog-backend Through Processing Faulty Locations and Entities	MEDIUM	NOT	SPBS-CR-22-08	4.1.7
Authentication of Unknown Users in Auth-Backend	CRITICAL	FULLY	SPBS-CR-22-09	4.1.8
Account Hijacking Due to Support of Multiple Authentication Providers	CRITICAL	FULLY	SPBS-CR-22-10	4.1.9
Execution of Arbitrary Docker Images in Scaffolder	MEDIUM	FULLY	SPBS-CR-22-11	4.1.10
Arbitrary Writes to File System of techdocs-backend	MEDIUM	FULLY	SPBS-CR-22-12	4.1.11
DoS Situation Due to multiple Techdoc Builds	LOW	FULLY	SPBS-CR-22-13	4.1.12
Log Injection	NONE	NOT	SPBS-CR-22-100	4.2.1
Missing Archive Validation Leads to Potential Remote Code Execution	NONE	FULLY	SPBS-CR-22-101	4.2.2
Copying of Workspace to Arbitrary File System Locations	NONE	NOT	SPBS-CR-22-102	4.2.3
No Authentication Required for scaffolder-backend	NONE	NOT	SPBS-CR-22-103	4.2.4
Wrong Age Check for Server Certificate	NONE	NOT	SPBS-CR-22-104	4.2.5
No Check Whether File for Location Spec Exists on Adding Them	NONE	FULLY	SPBS-CR-22-105	4.2.6
Yarn Installation Instructions Deprecated	NONE	NOT	SPBS-CR-22-106	4.2.7
nvm Installation Unverified	NONE	NOT	SPBS-CR-22-107	4.2.8
Use of Privileged Docker User	NONE	NOT	SPBS-CR-22-108	4.2.9
Docker Best Practices Not Applied	NONE	NOT	SPBS-CR-22-109	4.2.10
Expired Public Key Cleanup Only on Listing Public Keys	NONE	NOT	SPBS-CR-22-110	4.2.11
Missing Operational Security Guidelines	NONE	NOT	SPBS-CR-22-111	4.2.12
Insecure Kubernetes Examples in contrib Folder	NONE	NOT	SPBS-CR-22-112	4.2.13
Recommended Firewalling of Back-End	NONE	PARTLY	SPBS-CR-22-113	4.2.14
Missing/Broken Authentication for Ancestry Endoint in catalog-backend	NONE	FULLY	SPBS-CR-22-114	4.2.15

Table 2.1: Security-Relevant Findings

2.3 Scope

The aim of this test is to answer following key questions:

1. Is the core code safe from the attack vectors identified in the threat model?
2. What processes can the Backstage team adopt to screen plugins for security issues?
3. Are the project's build and deploy systems safe from intrusion? Can policies and procedures be improved?
4. Does backstage have sufficient documentation to prevent a junior admin from making serious security mistakes when setting up a backstage project? Are the default settings generally secure?
5. Is backstage's complex auth system robust and resilient against intrusion?

For the code review, the scope consisted mainly of the latest stable release of the main repository on 2022-03-03:

`https://github.com/backstage/backstage/tree/v0.70.0`

The retest was performed against version 1.4.0 of Backstage and a list of commits was provided to the testers.

The following five most important plugins for Backstage were included in the source code review:

- *plugins/auth-backend*
- *plugins/search-backend*
- *plugins/catalog-backend*
- *plugins/scaffolder-backend*
- TechDocs, consisting of:
  - *plugins/techdocs*
  - *plugins/techdocs-backend*
  - *packages/techdocs-common*

Finally, technical documentation to set up Backstage is in scope, located under:

`https://backstage.io/docs/`

2.4 Coverage

A security assessment attempts to find the most important or sometimes as many of the existing problems as possible, though it is practically never possible to rule out the possibility of additional weaknesses being found in the future.

The time allocated to X41 for this assessment was sufficient to yield a reasonable coverage of the given scope. As requested by the Backstage team, the review of the build system was replaced by a retest of the findings made during the source code review.

In particular the following questions as part of the scope definition were handled:

2.4.1 Is the core code safe from the attack vectors identified in the threat model?

X41 inspected the code for the security defects defined in the threat model and found several defects, two of them critical. The security relevant issues found are described in section 4.1.

2.4.2 What processes can the Backstage team adopt to screen plugins for security issues?

X41 discussed the security efforts and general security strategy regarding the code base and plugins with the Backstage team. Recommendations for further tests and actions are given in section 2.5 and side finding 4.2.12.

2.4.3 Are the project's build and deploy systems safe from intrusion? Can policies and procedures be improved?

This question has not been addressed and X41 was asked by the Backstage team to perform a retest of the findings instead.

2.4.4 Does backstage have sufficient documentation to prevent a junior admin from making serious security mistakes when setting up a backstage project? Are the default settings generally secure?

X41 reviewed the documentation and found it mostly sufficient in detail. As an improvement, notes about potential pitfalls and additionally needed security efforts were documented.

2.4.5 Is backstage's complex auth system robust and resilient against intrusion?

The authentication system offers a permission system to restrict access to resources. The implementation of this system was found to be correctly enforcing permissions, if they were enabled. Security issues might arise when a plugin is not applying checks for the permissions correctly and plugins need to be adapted in case new permissions are added. It is recommended to make the permission system more fail safe and add warning notes about potential pitfalls in the documentation for plugin developers.

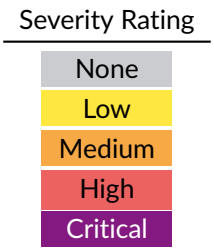
2.5 Recommended Further Tests

It is recommended to establish a process for code regular reviewing of plugins.

3 Rating Methodology for Security Vulnerabilities

Security vulnerabilities are given a purely technical rating by the testers as they are discovered during the test. Business factors and financial risks for the Backstage team are beyond the scope of a penetration test which focuses entirely on technical factors. Yet technical results from a penetration test may be an integral part of a general risk assessment. A penetration test is based on a limited time frame and only covers vulnerabilities and security issues which have been found in the given time, there is no claim for full coverage.

In total, five different ratings exist, which are as follows:



A low rating indicates that the vulnerability is either very hard for an attacker to exploit due to special circumstances, or that the impact of exploitation is limited, whereas findings with a medium rating are more likely to be exploited or have a higher impact. High and critical ratings are assigned when the testers deem the prerequisites realistic or trivial and the impact significant or very significant.

Findings with the rating ‘none’ are called side findings and are related to security hardening, affect functionality, or other topics that are not directly related to security. X41 recommends to mitigate these issues as well, because they often become exploitable in the future. Doing so will strengthen the security of the system and is recommended for defense in depth.

3.1 Common Weakness Enumeration

The CWE[1] is a set of software weaknesses that allows the categorization of vulnerabilities and weaknesses in software. If applicable, X41 provides the CWE-ID for each vulnerability that is discovered during a test.

CWE is a very powerful method to categorize a vulnerability and to give general descriptions and solution advice on recurring vulnerability types. CWE is developed by *MITRE*[2]. More information can be found on the CWE website at `https://cwe.mitre.org/`.

[1] Common Weakness Enumeration
[2] `https://www.mitre.org`

4 Results

This chapter describes the results of this test. The security-relevant findings are documented in Section 4.1. Additionally, findings without a direct security impact are documented in Section 4.2.

4.1 Findings

The following subsections describe findings with a direct security impact that were discovered during the test.

4.1.1 SPBS-CR-22-01: No Expiration Claim for JWT Access Token

Severity:	MEDIUM
Remediation:	FULLY
CWE:	613 – Insufficient Session Expiration
Affected Component:	packages/backend-common/src/tokens/ServerTokenManager.ts:getToken()

4.1.1.1 Remediation

The JWT[1] are generated with an expiration time set to `TOKEN_EXPIRY_AFTER`, which is one hour after the generation. This resolves this issue. The corresponding pull requests are 11262[2] and 11807[3].

[1] JSON Web Token
[2] `https://github.com/backstage/backstage/pull/11262`
[3] `https://github.com/backstage/backstage/pull/11807`

4.1.1.2 Description

During a source code review of the backend-common package it was noticed that the *ServerTokenManager.ts* file generates JWT access tokens. These access tokens, however, do not contain any expiration claim. An access token therefore stays valid indefinitely.

An attacker who obtains such an access token is able to impersonate the victim at any time, thereby obtaining access without any time restriction by using this access token. This potentially results in information leakages, or other unauthorized actions invoked by the attacker.

The code snippet 4.1 highlights the issue. It is evident that the JWT token does not contain any expiration claim.

```
async getToken(): Promise<{ token: string }> {
    const jwt = JWT.sign({ sub: 'backstage-server' }, this.signingKey, {
        algorithm: 'HS256',
    });

    return { token: jwt };
}
```

Listing 4.1: JWT Token Generation without Expiration

4.1.1.3 Solution Advice

X41 recommends to put an expiration claim into the JWT such that access tokens expire. This effectively prevents an attacker from impersonating the victim for an unlimited amount of time.

4.1.2 SPBS-CR-22-02: Constant Subject Claim for JWT Access Token

Severity:	HIGH
Remediation:	NOT
CWE:	None
Affected Component:	packages/backend-common/src/tokens/ServerTokenManager.ts:getToken()

4.1.2.1 Remediation

This issue was not addressed, since there is currently no concept of different server identities.

4.1.2.2 Customer Response

> "We have left this as a future feature to be implemented as we do not currently have the concepts of multiple different server identities."

4.1.2.3 Description

During a source code review of the backend-common package it was observed that the *ServerTokenManager.ts* generates a JWT access token. This access token uses a constant subject claim. Usually, services generate JWT access tokens after successful authentication of a user or another service, and put an identifier of either the user or service into a so-called subject claim.

Having a subject claim as part of a token is vital to identify the caller of service actions. Here, services which rely on the token generated by *ServerTokenManager.ts* can not identify the caller in a secure way since all tokens show `backstage-server` as subject.

This, together with Finding 4.1.1, results in a constant token which is identical for all users and services which use the *ServerTokenManager.ts*. If an attacker obtains such a JWT access token, the attacker is able to impersonate all users and services which use *ServerTokenManager.ts* for JWT token verification.

Code snippet 4.2 highlights the issue. Evidently, all tokens use the subject `backstage-server`.

```
1 async getToken(): Promise<{ token: string }> {
2         const jwt = JWT.sign({ sub: 'backstage-server' }, this.signingKey, {
3               algorithm: 'HS256',
4         });
```

```
5
6       return { token: jwt };
7   }
```

Listing 4.2: JWT Token Generation

4.1.2.4 Solution Advice

X41 recommends to put an identifier of the authenticated user or service as subject claim of the generated JWT access token.

4.1.3 SPBS-CR-22-03: Missing Rate Limiting for Backend Components

Severity:	LOW
Remediation:	NOT
CWE:	799 – Improper Control of Interaction Frequency
Affected Component:	plugins/*

4.1.3.1 Remediation

This was not resolved. The developers suggest that rate-limiting should be implemented by adopters.

4.1.3.2 Customer Response

> "Rate-limiting currently needs to be implemented by adopters if they need it, we do not have it built-in."

4.1.3.3 Description

The backend services do not implement any rate limiting or throttling mechanism.

This enables an attacker to run so-called DoS[4] attacks on the backend services. Such attacks either crash entire services, or disrupt legitimate users from being able to consume the services.

For illustration purposes, the code snippet 4.3 shows one endpoint within the *plugins/scaffolder-backend/src/service/router.ts* without rate limiting in the scaffolder-backend plugin. The developers further clarified that there are no other protections with regards to rate limiting in place.

```
1  .get('/v2/actions', async (_req, res) => {
2            const actionsList = actionRegistry.list().map(action => {
3              return {
4                    id: action.id,
5                    description: action.description,
6                    schema: action.schema,
7              };
8            });
9            res.json(actionsList);
10 })
```

[4] Denial of Service

Listing 4.3: Router Endpoint without Rate Limiting

4.1.3.4 Solution Advice

X41 recommends to implement rate limiting or throttling techniques for all endpoints to prevent resource exhaustion and disruption of service for legitimate users.

4.1.4 SPBS-CR-22-04: Unauthenticated DoS Situation Due to Scaffolder Event Streaming

Severity:	MEDIUM
Remediation:	NOT
CWE:	400 – Uncontrolled Resource Consumption ('Resource Exhaustion')
Affected Component:	plugins/scaffolder-backend/src/service/router.ts

4.1.4.1 Remediation

This was not resolved. The developers suggest that rate-limiting should be implemented by adopters.

4.1.4.2 Customer Response

Please see the response for finding *SPBS-CR-22-03* in subsection 4.1.3.2.

4.1.4.3 Description

The scaffolder-backend supports the streaming of events which occur during the execution of tasks. For that purpose, a client provides the identifier for the task and the scaffolder-backend streams events towards the client.

The client does not need to authenticate to consume the event stream and the scaffolder-backend does not limit the number of concurrent event streams.

An attacker could leverage this and open many event streams at the same time without authenticating to the backend. Each stream consumes resources, hence the attacker could exhaust the resources on the scaffolder-backend, thereby resulting in a DoS situation. Code snippet 4.4 shows the issue.

```
1 .get('/v2/tasks/:taskId/eventstream', async (req, res) => {
2       const { taskId } = req.params;
3       const after =
4         req.query.after !== undefined ? Number(req.query.after) : undefined;
5
6       logger.debug(`Event stream observing taskId '${taskId}' opened`);
7
8       // Mandatory headers and http status to keep connection open
```

```
      res.writeHead(200, {
        Connection: 'keep-alive',
        'Cache-Control': 'no-cache',
        'Content-Type': 'text/event-stream',
      });

      // After client opens connection send all events as string
      const subscription = taskBroker.event$({ taskId, after }).subscribe({
        error: error => {
          logger.error(
            `Received error from event stream when observing taskId '${taskId}', ${error}`,
          );
        },
        next: ({ events }) => {
          let shouldUnsubscribe = false;
          for (const event of events) {
            res.write(
              `event: ${event.type}\ndata: ${JSON.stringify(event)}\n\n`,
            );
            if (event.type === 'completion') {
              shouldUnsubscribe = true;
            }
          }
          // res.flush() is only available with the compression middleware
          res.flush?.();
          if (shouldUnsubscribe) subscription.unsubscribe();
        },
      });

      // When client closes connection we update the clients list
      // avoiding the disconnected one
      req.on('close', () => {
        subscription.unsubscribe();
        logger.debug(`Event stream observing taskId '${taskId}' closed`);
      });
})
```

Listing 4.4: Unbound Event Streaming

4.1.4.4 Solution Advice

X41 recommends to allow only authenticated users to access the event streaming and limit the number of concurrent event streams. The scaffolder-backend should provide a task token in terms of a JWT access token upon executing a new task, consequently checking this token on accessing information regarding this task.

4.1.5 SPBS-CR-22-05: Information Leakage through Misconfiguration via Templates

Severity:	LOW
Remediation:	FULLY
CWE:	200 – Exposure of Sensitive Information to an Unauthorized Actor
Affected Component:	plugins/scaffolder-backend/src/scaffolder/actions/builtin/fetch/helpers.ts

4.1.5.1 Remediation

This was resolved via pull request 12001[5], which limits the file templates to URL[6] locations.

4.1.5.2 Description

During a source code review of the scaffolder-backend it was noticed that the backend offers so-called actions. The scaffolder-backend executes these actions after loading the corresponding template of the action. There are several built-in actions available, including also so-called fetch actions. These actions use the *plugins/scaffolder-backend/src/scaffolder/actions/builtin/fetch/helpers.ts* file, which copies files from the base directory of the template to a provided fetch directory (as input). The function checks whether the fetch directory is a child of the template's base directory, and stops the execution of the action if this is not the case. The base directory of a template is provided implicitly by the operator on registering new templates either in terms of its file location, or as a URL.

If an operator places such templates in the root directory of the file system, all provided fetch directories will be considered valid. This would allow an attacker to read from arbitrary file locations of the scaffolder-backend host, which could leak sensitive information. However, it needs to be emphazied that it is really fully up to the operator from where these templates are provided, which tremendously lowers the chance of a successful exploitation.

Code Listing 4.5 shows the issue. The base directory, referred to also as `baseUrl` in the source code, is used to resolve the source directory in a safe manner. If the base directory points to /, arbitrary file system reads are possible.

```
1 if (!fetchUrlIsAbsolute && baseUrl?.startsWith('file://')) {
2     const basePath = baseUrl.slice('file://'.length);
3     const srcDir = resolveSafeChildPath(path.dirname(basePath), fetchUrl);
```

[5] https://github.com/backstage/backstage/pull/12001
[6] Uniform Resource Locator

```
4       await fs.copy(srcDir, outputPath);
5   } else {
```

Listing 4.5: Arbitrary File Reads by Improper Templates File Placement

4.1.5.3 Solution Advice

X41 recommends to configure a path, or list of paths, under which all base directories must reside.

4.1.6 SPBS-CR-22-06: Potential Information Leakage through Symlinks in Repositories

Severity:	HIGH
Remediation:	FULLY
CWE:	200 – Exposure of Sensitive Information to an Unauthorized Actor
Affected Component:	plugins/scaffolder-backend/src/scaffolder/actions/builtin/publish/gitHubPullRequest.ts, plugins/scaffolder-backend/src/scaffolder/actions/builtin/publish/gitlabMergeRequest.ts

4.1.6.1 Remediation

The issue was addressed in pull request *12511*[7] by verifying that a target file is not a symbolic link. The way the check is implemented is not safe against race conditions, where a local attacker with file system level privileges on the file being verified could try to replace the file with a symbolic link after the check was passed, but before usage happens (TOCTOU[8]). Assuming that the process doing the verification is running as a dedicated, trustworthy user, the mitigation can be assumed to be working and the issue thus fully mitigated.

4.1.6.2 Description

The backend provides built-in actions for GitHub Pull Requests and GitLab Merge Requests. Both actions read the file contents from a repository into memory. The GitHub Pull Request creates a ZIP archive from the file contents whereas the GitLab Merge Request pushes an action into a list for each file.

Git repositories allow for the use of symlinks inside repositories. Specifically, given that the symlink destination file exists on the host, Git preserves the symlink via the repository. Neither implementation checks whether a file of a repository symlinks to a file outside of the repository or workspace directory.

This allows an attacker who is able to provide symlinks in a repository to obtain sensitive information from the host file system of the scaffolder-backend as part of a ZIP archive (for GitHub Pull Requests) or by reading out the destination of actions (for GitLab Merge Requests).

Code Listing 4.6 shows the issue for GitHub Pull Requests. The file contents from symlinked files

[7] `https://github.com/backstage/backstage/pull/12511/commits/5185c58bf5461933f4ab1ef628e4637c99f18370`

[8] Time-of-check to time-of-use

end up in a ZIP archive.

```
const fileRoot = sourcePath
        ? resolveSafeChildPath(ctx.workspacePath, sourcePath)
        : ctx.workspacePath;

const localFilePaths = await globby(['./**', './**/.*', '!.git'], {
    cwd: fileRoot,
        gitignore: true,
        dot: true,
});

const fileContents = await Promise.all(
        localFilePaths.map(filePath => {
          const absPath = resolveSafeChildPath(fileRoot, filePath);
          const base64EncodedContent = fs
            .readFileSync(absPath)
            .toString('base64');
          [...]
      const encoding: Encoding = 'base64';
          return {
            encoding: encoding,
            content: base64EncodedContent,
            mode: githubTreeItemMode,
          };
        }),
      );

const repoFilePaths = localFilePaths.map(repoFilePath => {
        return targetPath ? `${targetPath}/${repoFilePath}` : repoFilePath;
});

const changes = [
        {
          files: zipObject(repoFilePaths, fileContents),
          commit: title,
        },
];
```

Listing 4.6: Potential Information Leakage through GitHub Pull Request Action

Code Listing 4.7 shows the issue for GitLab Merge Requests. It is evident that the file contents from symlinked files end up in the actions list of the GitLab Merge Request.

```
const fileRoot = ctx.workspacePath;
const localFilePaths = await globby([`${ctx.input.targetPath}/**`], {
        cwd: fileRoot,
```

```
        gitignore: true,
        dot: true,
});

const fileContents = await Promise.all(
        localFilePaths.map(p => readFile(resolveSafeChildPath(fileRoot, p))),
);

const repoFilePaths = localFilePaths.map(repoFilePath => {
        return repoFilePath;
});

for (let i = 0; i < repoFilePaths.length; i++) {
        actions.push({
          action: 'create',
          filePath: repoFilePaths[i],
          content: fileContents[i].toString(),
        });
}
```

Listing 4.7: Potential Information Leakage through GitLab Merge Request Action

4.1.6.3 Solution Advice

X41 recommends to disable symlinks which point outside repositories.

4.1.7 SPBS-CR-22-08: DoS Situation for catalog-backend Through Processing Faulty Locations and Entities

Severity:	MEDIUM
Remediation:	NOT
CWE:	400 – Uncontrolled Resource Consumption ('Resource Exhaustion')
Affected Component:	plugins/catalog-backend/src/processing/DefaultCatalogProcessingEngine.ts

4.1.7.1 Remediation

This issue was not resolved as it is intended behavior. For the future it is planned to add more observability that allows for monitoring of catalog processing latency.

4.1.7.2 Customer Response

> "This is the intended behavior of the catalog. We will likely add observability for this case in the future though, so that it can be detected more easily. There are already metrics for keeping an eye on the catalog processing latency."

4.1.7.3 Description

The catalog-backend processes location specs in the *DefaultCatalogProcessingEngine.ts* file. For that purpose it provides a ***loadTask()*** callback which loads only a defined number of unprocessed entities in one go. After loading them, the *DefaultCatalogProcessingEngine.ts* file processes the entities. It does not, however, delete entities from the database which lead to errors or are malformed.

This allows an attacker who has access to the catalog-backend and is able to specify faulty entities or faulty locations to bring the catalog-backend into a DoS situation. An attacker could add locations containing faulty entities, which would pile up in the catalog-backend due to a missing delete functionality on processing them. Consequently, as also confirmed by the developers, the catalog-backend will continue to try to process the faulty entities, which essentially blocks legitimate locations and entities from being processed.

Code Listing 4.8 shows the issue. It is evident that faulty entities stay within the catalog-backend and are not deleted by the backend.

```
if (!result.ok) {
    await this.processingDatabase.transaction(async tx => {
    await this.processingDatabase.updateProcessedEntityErrors(tx, {
        id,
        errors: errorsString,
        resultHash,
    });
    });
    await this.stitcher.stitch(
    new Set([stringifyEntityRef(unprocessedEntity)]),
    );
    track.markSuccessfulWithErrors();
    return;
}
```

Listing 4.8: Faulty Entities Do Not Get Deleted from the Database.

4.1.7.4 Solution Advice

X41 recommends to delete faulty entities from the database.

4.1.8 SPBS-CR-22-09: Authentication of Unknown Users in Auth-Backend

Severity:	CRITICAL
Remediation:	FULLY
CWE:	303 – Incorrect Implementation of Authentication Algorithm
Affected Component:	plugins/auth-backend/src/providers/*

4.1.8.1 Remediation

This was addressed in pull request 10300[9] with additional adjustments to the documentation in 11901[10]. This resolves this issue.

4.1.8.2 Description

The auth-backend acts as a wrapper for the authentication methods which backstage supports. Backstage allows the following authentication providers:

1. atlassian
2. auth0
3. aws-alb
4. bitbucket
5. gcp-iap
6. github
7. gitlab
8. google
9. microsoft
10. oauth2
11. oauth2-proxy
12. oidc

[9] https://github.com/backstage/backstage/pull/10300
[10] https://github.com/backstage/backstage/pull/11901

13. okta
14. onelogin
15. saml

Most of them have a so-called ***defaultSignInResolver*** which converts from a user's authentication result of a provider to a Backstage identity. Those ***defaultSignInResolver***s, however, do not check whether the corresponding user of the authentication exists within the Backstage ecosystem or not.

This allows an attacker who is registered on one of the supported authentication platforms to sign in to the auth-backend even though Backstage does not know the user.

Code Listing 4.9 shows the issue for the Auth0 authentication provider. It is evident from this listing that this provider does not check whether the successfully authenticated user exists in the Backstage ecosystem or not. Similar functions exist also for most of the other authentication providers.

```
const defaultSignInResolver: SignInResolver<OAuthResult> = async info => {
  const { profile } = info;

  if (!profile.email) {
    throw new Error('Profile does not contain an email');
  }

  const id = profile.email.split('@')[0];

  return { id, token: '' };
};
```

Listing 4.9: Default Sign-In Resolver for the Auth0 Provider

4.1.8.3 Solution Advice

X41 recommends to always ensure that successfully authenticated users exist in the Backstage ecosystem.

4.1.9 SPBS-CR-22-10: Account Hijacking Due to Support of Multiple Authentication Providers

Severity:	CRITICAL
Remediation:	FULLY
CWE:	287 – Improper Authentication
Affected Component:	plugins/auth-backend/src/providers/*

4.1.9.1 Remediation

This was addressed in pull request 10300[11] with additional adjustments to the documentation in 11901[12]. This resolves this issue.

4.1.9.2 Description

The auth-backend acts as a wrapper for the authentication methods which backstage supports. Backstage allows the following authentication providers:

1. atlassian
2. auth0
3. aws-alb
4. bitbucket
5. gcp-iap
6. github
7. gitlab
8. google
9. microsoft
10. oauth2
11. oauth2-proxy
12. oidc

[11] https://github.com/backstage/backstage/pull/10300
[12] https://github.com/backstage/backstage/pull/11901

13. okta
14. onelogin
15. saml

The auth-backend defers the authentication of a user to these external providers. After successful authentication the auth-backend associates a backstage identity with the user by extracting the username from the email address of the credentials. Furthermore, the auth-backend does not keep track of allowed authentication providers for users or organizations but rather enables users to use all of the aforementioned providers.

This enables an attacker to mount several kinds of attacks, including the following:

- In case the victim of the attacker does not have an account on one of the supported authentication providers, an attacker could create an email address with the name of the email address matching the victims username and sign-in as the victim to Backstage.
- In case the victim does have accounts within several supported authentication providers, it suffices for the attacker to hijack one of them to impersonate the victim within the Backstage ecosystem.

Code Listing 4.10 shows the extraction of the backstage identity from the result of a successful authentication of a user for the Auth0 authentication provider. It is obvious that the auth-backend extracts the user identity for Backstage from the email address of the user presented on authentication. Further, the auth-backend does not check whether the user identified by the name extracted from the local part of the email address is allowed to use this authentication provider or not.

```
1  const defaultSignInResolver: SignInResolver<OAuthResult> = async info => {
2    const { profile } = info;
3
4    if (!profile.email) {
5      throw new Error('Profile does not contain an email');
6    }
7
8    const id = profile.email.split('@')[0];
9
10   return { id, token: '' };
11 };
```

Listing 4.10: Extraction of the Backstage Identity From the Users Email Address

4.1.9.3 Solution Advice

X41 recommends to implement an allow-list approach for authentication providers at the level of organizations, but also at the level of individual users. Furthermore, the auth-backend should use the full email address used for authentication as the identity of a Backstage user.

4.1.10 SPBS-CR-22-11: Execution of Arbitrary Docker Images in Scaffolder

Severity:	MEDIUM
Remediation:	FULLY
CWE:	None
Affected Component:	plugins/scaffolder-backend-module-rails/src/actions/fetch/rails/rails-NewRunner.ts

4.1.10.1 Remediation

This was addressed in pull request 11254[13], which verifies the supplied image name against an allow-list.

4.1.10.2 Description

The scaffolder-backend indirectly supports, via a module for Rails, the execution of Docker images. In this particular case, the scaffolder-backend uses the `rails` command for Ruby on Rails applications. In case the `rails` command is not available on the scaffolder-backend host, the plugin executes the `rails` command inside a Docker container. For that purpose, the implementation requires an image name to identify the Docker image which needs to be run. The scaffolder-backend module for Rails does not verify the provided image name.

This enables an attacker to who is able to invoke this action to provide arbitrary image names to the rails module of the scaffolder-backend. This in turn allows for the execution of arbitrary Docker containers on the scaffolder-backend.

Code Listing 4.11 shows the issue. It is evident that the function does not check the image name against an allowed list of images.

```
1  public async run({
2      workspacePath,
3      values,
4      logStream,
5    }: {
6      workspacePath: string;
7      values: JsonObject;
8      logStream: Writable;
9    }): Promise<void> {
10     const intermediateDir = path.join(workspacePath, 'intermediate');
```

[13] https://github.com/backstage/backstage/pull/11254

```
    await fs.ensureDir(intermediateDir);
    const resultDir = path.join(workspacePath, 'result');

    const { name, imageName, railsArguments } = values;
    [...]
    const baseCommand = 'rails';
    const baseArguments = ['new'];
    const commandExistsToRun = await commandExists(baseCommand);

    if (commandExistsToRun) {
      [...]
    } else {
      const arrayExtraArguments = railsArgumentResolver(
        '/input',
        railsArguments as RailsRunOptions,
        true,
      );
      await this.containerRunner.runContainer({
        imageName: imageName as string,
        command: baseCommand,
        args: [...baseArguments, `/output/${name}`, ...arrayExtraArguments],
        mountDirs,
        workingDir: '/input',
        // Set the home directory inside the container as something that applications can
        // write to, otherwise they will just fail trying to write to /
        envVars: { HOME: '/tmp' },
        logStream,
      });
    }
    [...]
}
```

Listing 4.11: Execution of Arbitrary Docker Containers

4.1.10.3 Solution Advice

X41 recommends to follow an allow-list approach of images to be executed as part of the rails module for the scaffolder-backend.

4.1.11 SPBS-CR-22-12: Arbitrary Writes to File System of techdocs-backend

Severity:	MEDIUM
Remediation:	FULLY
CWE:	None
Affected Component:	packages/techdocs-common/src/stages/publish/local.ts

4.1.11.1 Remediation

This was resolved in commit 5296b00[14], which safely resolves the paths.

4.1.11.2 Description

The techdocs-backend implements a build process to create techdocs by using mkdocs build. For that purpose the techdocs-backend follows a three step process, comprised of the prepare, generate and publish steps. Backstage supports several different publish actions, one of them being a *local* action. The *local* publish action copies the output of the generate step to the local file system of the techdocs-backend. The destination path of this copy is compiled from the *namespace*, the *kind* and the *name* of the techdoc, without sanitizing the individual parts from characters like '..' or '/'. Further, it shall be noted that the *local* publish action is the default publish action.

This allows an attacker who is able to create techdocs to provide crafted values for *namespace*, *kind* and *name* to write the output of the generate techdocs build step to arbitrary file locations.

Code Listing 4.12 shows the issue. It is evident that the local publish action does not escape or sanitize the individual parts comprising the final path.

```
protected staticEntityPathJoin(...allParts: string[]): string {
  if (this.legacyPathCasing) {
    const [namespace, kind, name, ...parts] = allParts;
    return path.join(staticDocsDir, namespace, kind, name, ...parts);
  }
  const [namespace, kind, name, ...parts] = allParts;
  return path.join(
    staticDocsDir,
    namespace.toLowerCase(),
    kind.toLowerCase(),
    name.toLowerCase(),
    ...parts,
```

[14] https://github.com/backstage/backstage/commit/5296b00d4a8c7b7079ccb11b27eb1376e40b8125

```
13     );
14   }
```

Listing 4.12: Path Compilation for the Destination Directory of Techdoc Publish

4.1.11.3 Solution Advice

X41 recommends to check and sanitize the values of *namespace*, *kind* and *name* of a techdoc before compiling the destination path of the local publish action.

4.1.12 SPBS-CR-22-13: DoS Situation Due to multiple Techdoc Builds

Severity:	LOW
Remediation:	FULLY
CWE:	400 – Uncontrolled Resource Consumption ('Resource Exhaustion')
Affected Component:	packages/backend-common/src/util/DockerContainerRunner.ts

4.1.12.1 Remediation

The number of techdoc builds was limited to a default of 10 concurrent builds at the same time in pull request 11816[15]. This resolves the issue.

4.1.12.2 Description

The techdocs-backend implements a build process to create techdocs by using `mkdocs build`. For that purpose the techdocs-backend follows a three step process, comprised of a prepare, generate and publish step. For the generate step two options are available: local execution of the `mkdocs build` command on the host of the techdocs-backend, or via a dedicated Docker container running the `mkdocs build` command. The latter is done by using the *spotify/techdocs* image from Docker Hub, which has a size of roughly 460MB. When using Docker containers for techdoc builds, the techdocs-backend spawns a new Docker container for each build process from the *spotify/techdocs* image. The number of concurrent techdoc builds is not upper bounded and thereby vulnerable to resource exhaustion, due to too many Docker containers running at the same time.

This allows an attacker who is able to trigger techdoc builds in the techdocs-backend to run a DoS attack by requesting multiple techdoc builds at the same time. The techdocs-backend spawns a new Docker container for each build request, thereby consuming all the memory of the host.

Code Listing 4.13 shows the issue. The number of concurrent Docker containers is not upper bounded.

```
1 async runContainer(options: RunContainerOptions) {
2   const {
3     imageName,
4     command,
5     args,
6     logStream = new PassThrough(),
```

[15] https://github.com/backstage/backstage/pull/11816

```
7       mountDirs = {},
8       workingDir,
9       envVars = {},
10      pullImage = true,
11    } = options;
12    [...]
13    const [{ Error: error, StatusCode: statusCode }] =
14      await this.dockerClient.run(imageName, args, logStream, {
15        Volumes,
16        HostConfig: {
17          AutoRemove: true,
18          Binds,
19        },
20        ...(workingDir ? { WorkingDir: workingDir } : {}),
21        Entrypoint: command,
22        Env,
23        ...userOptions,
24      } as Docker.ContainerCreateOptions);
25    [...]
26  }
```

Listing 4.13: The Number of Docker Containers Running is Unbounded.

4.1.12.3 Solution Advice

X41 recommends to limit the number of concurrent techdoc build processes on the techdocs-backend.

4.2 Side Findings

The following observations do not have a direct security impact, but are related to security hardening, affect functionality, or other topics that are not directly related to security. X41 recommends to mitigate these issues as well, because they often become exploitable in the future. Doing so will strengthen the security of the system and is recommended for defense in depth.

4.2.1 SPBS-CR-22-100: Log Injection

Affected Component:	WebSocket Endpoint Logging
Remediation:	NOT

4.2.1.1 Remediation

This will be addressed in the future, but is deemed low priority since the production servers usually use JSON[16] output where this issue does not occur.

4.2.1.2 Customer Response

> "In the production configuration we use JSON formatted log output, which escapes appropriately. This is not enabled by default in our Dockerfiles, so we'll fix that."

4.2.1.3 Description

The WebSocket endpoint on TCP[17] port 7007 logs data about incoming requests to *stdout* without further escaping the variables provided.

```
1 GET /api/catalog/entities?a="<--bad HTTP/1.1
2 Host: localhost:7007
3 Accept-Encoding: gzip, deflate
4 Accept: */*
5 Accept-Language: en
6 User-Agent: in" jec" tion"
7 Connection: close
```

[16] JavaScript Object Notation
[17] Transmission Control Protocol

Listing 4.14: Request to TCP 7007

The output is separated by spaces with some sections being quoted. The URI[18] component allows quotes to be sent into the log whereas the user-agent HTTP[19] header allows both quotes and spaces to be injected.

```
1 [1] 2022-03-04T07:46:28.394Z backstage info ::ffff:127.0.0.1 - - [04/Mar/2022:07:46:28 +0000]
  ↪ "GET /api/catalog/entities?a="<--bad HTTP/1.1" 200 - "-" "in" jec" tion"" type=incomingRequest
```

Listing 4.15: Log Output for Injection Request

This might confuse parsers handling the logging data and could potentially be abused by attackers.

4.2.1.4 Solution Advice

X41 suggests to escape characters that are used in the output format structure.

[18] Uniform Resource Identifier
[19] HyperText Transfer Protocol

4.2.2 SPBS-CR-22-101: Missing Archive Validation Leads to Potential Remote Code Execution

Affected Component:	backend-common
Remediation:	FULLY

4.2.2.1 Remediation

This is handled by the libraries and mitigted in-depth by pull request 13054[20].

4.2.2.2 Description

The backend-common repository implements read tree response readers within the folder *packages/backend-common/src/reading/tree*. The purpose of the implementation in this folder is to unpack the response of a trusted repository provider. The formats which the source code currently supports are ZIP, tar and so-called readable arrays. The ZIP format was known to be vulnerable to so-called ZIP-Slip attacks, in which an attacker provides a crafted ZIP file containing file entries like *../../evil.sh*, which allow an attacker to escape from the directory that is the target of the unzip operation.

The unzipper package used to perform the unzip operation for the source code in-scope uses a version in which this vulnerability is fixed. TAR archives are also known to be vulnerable to this kind of attack and it seems that this vulnerability has not been closed yet in packages. Furthermore, the readable array response implementation, as shown in Code Snippet 4.16 by showing parts of *packages/backend-common/src/reading/tree/ReadableArrayResponse.ts*, also does not properly validate the provided paths. This in turn allows an attacker to escape from the current directory, and perform writes outside of the target directory. If the archives contain scripts, this can lead to remote code execution.

Since all providers at the moment are trusted, the complexity of such an attack is high. In case the functionality is used together with untrusted source providers in the future, the attack complexity would drop, accordingly.

```
1 async dir(options?: ReadTreeResponseDirOptions): Promise<string> {
2     this.onlyOnce();
3
4     const dir =
5         options?.targetDir ??
```

[20] https://github.com/backstage/backstage/pull/13054

```
6       (await fs.mkdtemp(platformPath.join(this.workDir, 'backstage-')));
7
8       for (let i = 0; i < this.stream.length; i++) {
9           if (!this.stream[i].path.endsWith('/')) {
10              await pipeline(
11              this.stream[i].data,
12              fs.createWriteStream(
13              platformPath.join(dir, basename(this.stream[i].path)),
14              ),
15              );
16          }
17      }
18
19      return dir;
20  }
```

Listing 4.16: Log Output for Injection Request

4.2.2.3 Solution Advice

X41 suggests to sanitize all paths before writing to the file system.

4.2.3 SPBS-CR-22-102: Copying of Workspace to Arbitrary File System Locations

Affected Component:	scaffolder-backend
Remediation:	NOT

4.2.3.1 Remediation

This issue is not resolved. The feature will be deprecated and removed in the near future.

4.2.3.2 Customer Response

> "This action is only used for local development and is not installed by default. We will however deprecate and remove the actions to avoid any risk, as there are now other more convenient options to use instead."

4.2.3.3 Description

The file *backstage-0.70.0/plugins/scaffolder-backend/src/scaffolder/actions/builtin/publish/file.ts*, which is part of the built-in publish actions, allows to copy the content of a workspace to arbitrary new file system locations. It needs to be emphasized that these actions are not installed by default and also the documentation warns the user about the security implications on using it.

However, an attacker who can trigger this action is able to copy the content of workspaces to arbitrary, new locations within the file system of the scaffolder-backend. Listing 4.17 shows the issue.

```
async handler(ctx) {
    const { path } = ctx.input;

          const exists = await fs.pathExists(path);
          if (exists) {
            throw new InputError('Output path already exists');
          }
          await fs.ensureDir(dirname(path));
          await fs.copy(ctx.workspacePath, path);
    },
```

Listing 4.17: Unverified Path Used

4.2.3.4 Solution Advice

X41 suggests to check that the provided path is a child path of the templates base directory before copying the entire workspace content.

4.2.4 SPBS-CR-22-103: No Authentication Required for scaffolder-backend

Affected Component:	scaffolder-backend
Remediation:	NOT

4.2.4.1 Remediation

This issue is not resolved.

4.2.4.2 Customer Response

> "This applies to all backend endpoints and fixing this will be part of a bigger milestone to add auth across all endpoints."

4.2.4.3 Description

During a review of the scaffolder-backend repository it was noticed that the backend offers several API endpoints. Most of them do not require any authentication. Specifically, the following endpoints do not require any authentication:

- `GET /v2/actions`
- `GET /v2/tasks/:taskId`
- `GET /v2/tasks/:taskId/eventstream`
- `GET /v2/tasks/:taskId/events`

Endpoints without authentication are visible and reachable by everyone. Therefore, if the endpoints provide sensitive information about task execution, or even secrets, an attacker could acquire such information easily.

4.2.4.4 Solution Advice

X41 suggests to protect all endpoints with authentication. This issue can not fully be resolved by firewalling, since it could still be abused with attacks such as SSRF[21].

[21] Server-Side Request Forgery

4.2.5 SPBS-CR-22-104: Wrong Age Check for Server Certificate

Affected Component:	packages/backend-common/src/service/lib/hostFactory.ts
Remediation:	NOT

4.2.5.1 Remediation

This issue is not resolved but will be addressed in the future.

4.2.5.2 Customer Response

> "Will be creating an issue for this on GitHub, as it's something that might just lead to inconvenience in local development."

4.2.5.3 Description

The *hostFactory.ts* file reads a certificate from the file system. Certificates have a well-defined structure, including a field indicating the expiry date of a certificate.

However, the current implementation does not evaluate this expiry date, but rather uses local file properties to determine whether the certificate is expired or not. Therefore, it is possible that an expired certificate gets mistakenly loaded and used, which would lead clients to abort connections due to an expired certificate.

Code listing 4.18 shows the issue. It is evident from the snippet that the file uses file properties to determine whether the certificate should be read or not.

```
1 let cert = undefined;
2 if (await fs.pathExists(certPath)) {
3     const stat = await fs.stat(certPath);
4     const ageMs = Date.now() - stat.ctimeMs;
5     if (stat.isFile() && ageMs < ALMOST_MONTH_IN_MS) {
6         cert = await fs.readFile(certPath);
7     }
8 }
```

Listing 4.18: Certificate Expiry Check

4.2.5.4 Solution Advice

X41 recommends to always read the certificate file data and check the expiration date provided by the certificate.

4.2.6 SPBS-CR-22-105: No Check Whether File for Location Spec Exists on Adding Them

Affected Component:	plugins/catalog-backend/src/modules/core/DefaultLocationStore.ts
Remediation:	FULLY

4.2.6.1 Remediation

This issue was resolved by pull request 12001[22].

4.2.6.2 Description

The *DefaultLocationStore.ts* file stores new location specs of the catalog backend to the database. There are different types of location specs which the backend supports, also including a file location spec. The code in the file *DefaultLocationStore.ts* does not check for file location specs whether the file exists or not. This in turn allows to add file specs pointing at files which do not exist.

Code listing 4.19 shows the issue. It is clear that the code in the *DefaultLocationStore.ts* file does not verify whether a file exists for file location specs.

```
async createLocation(input: LocationInput): Promise<Location> {
    const location = await this.db.transaction(async tx => {
      // Attempt to find a previous location matching the input
      const previousLocations = await this.locations(tx);
      // TODO: when location id's are a compilation of input target we can remove this full
      // lookup of locations first and just grab the by that instead.
      const previousLocation = previousLocations.some(
        l => input.type === l.type && input.target === l.target,
      );
      if (previousLocation) {
        throw new ConflictError(
          `Location ${input.type}:${input.target} already exists`,
        );
      }

      const inner: DbLocationsRow = {
        id: uuid(),
        type: input.type,
        target: input.target,
      };
```

[22] https://github.com/backstage/backstage/pull/12001

```
21
22        await tx<DbLocationsRow>('locations').insert(inner);
23
24        return inner;
25      });
26      const entity = locationSpecToLocationEntity(location);
27      await this.connection.applyMutation({
28        type: 'delta',
29        added: [{ entity, locationKey: getEntityLocationRef(entity) }],
30        removed: [],
31      });
32
33      return location;
34    }
```

Listing 4.19: Missing File Exists Check

4.2.6.3 Solution Advice

X41 recommends to verify for location specs of type file whether the file exists. In case the file does not exist, adding the location spec should fail.

4.2.7 SPBS-CR-22-106: Yarn Installation Instructions Deprecated

Affected Component:	https://backstage.io/docs/getting-started/
Remediation:	NOT

4.2.7.1 Remediation

This issue does not seem to be resolved.

4.2.7.2 Customer Response

> "These are the installation instructions for the yarn version that we use so they need to stay. We are in the process of moving to a more recent version though."

4.2.7.3 Description

The Getting Started page links to `https://classic.yarnpkg.com/en/docs/install` which shows a warning that this install method, for Yarn version 1, is not the latest version anymore as of January 2020 and that support will end at some undefined time. It is not stated how much warning time would be provided once this 'maintenance mode' period ends.

4.2.7.4 Solution Advice

X41 recommends to update the link to the latest installation instructions to ensure stable support for the installed version.

4.2.8 SPBS-CR-22-107: nvm Installation Unverified

Affected Component:	https://backstage.io/docs/getting-started/
Remediation:	NOT

4.2.8.1 Remediation

This issue does not seem to be resolved.

4.2.8.2 Customer Response

> "[We] strongly prefer NVM over binary installations of Node.js, since the binary installation does not set up proper file system permissions for executable files, leading to risky use of sudo. The installation procedure for NVM is in our opinion worth the tradeoff, and the installation instructions do hint and downloading the script instead too."

4.2.8.3 Description

The Getting Started page links to `https://github.com/nvm-sh/nvm#install--update-script` as the preferred method of installing `nvm` to get a Node.js Active LTS[23] release. Users are instructed to pipe curl output from `https://raw.githubusercontent.com/` into bash. This creates an implicit dependency on GitHub (Microsoft), who can modify the contents of this download generally, for `curl` user agents, or for specific targets. There are no checks to ensure that the executed code originates from the `nvm` developers.

Docker has a similar installation procedure, though using a download at `docker.com`. Ideally, something like a PGP[24] signature would be added so at least the more vigilant users could notice if the domain was compromised, but serving an installation script from one's own servers is a step up from directly depending on third-party hosting.

4.2.8.4 Solution Advice

X41 recommends to instead recommend the binary download method already mentioned on the Getting Started page. Alternatively, users could be made aware of the fact that nvm release tags

[23] Long Term Support
[24] Pretty Good Privacy

are signed. In that case, the git install [25] should be used instead of the install script.

[25] `https://github.com/nvm-sh/nvm#git-install`

4.2.9 SPBS-CR-22-108: Use of Privileged Docker User

Affected Component:	backstage/packages/backend-common/src/util/DockerContainer-Runner.ts
Remediation:	NOT

4.2.9.1 Remediation

The risk was accepted with the following developer comment:

4.2.9.2 Customer Response

> "Our understanding is that the code improves security, as we avoid running as the root user."

4.2.9.3 Description

The comment listed in listing 4.20 mentions that, since root is needed for the used configuration on Mac systems, it is also used in Linux. So a limitation found predominantly on developer machines (Mac) leads to a decrease in security hardening on production systems (Linux).

```
1 // Files that are created inside the Docker container will be owned by
2 // root on the host system on non Mac systems, because of reasons. Mainly the fact that
3 // volume sharing is done using NFS on Mac and actual mounts in Linux world.
4 // So we set the user in the container as the same user and group id as the host.
5 // On Windows we don't have process.getuid nor process.getgid
```

Listing 4.20: Use of Privileged Docker User

4.2.9.4 Solution Advice

X41 recommends to not relax security hardening for production systems.

4.2.10 SPBS-CR-22-109: Docker Best Practices Not Applied

Affected Component:	Docker
Remediation:	NOT

4.2.10.1 Remediation

This issue does not seem to be resolved. But best practices will be applied in the future.

4.2.10.2 Customer Response

> "We will try to update these."

4.2.10.3 Description

The provided Dockerfiles violate some best practices, listed below:

- *contrib/docker/cookiecutter-with-jinja2-extensions/Dockerfile*
  - missing USER directive
- *contrib/docker/kubernetes-example-backend/Dockerfile*
  - missing USER directive
  - Python is installed using curl, without any signature verification
- *contrib/docker/devops/Dockerfile*
  - missing USER directive

4.2.10.4 Solution Advice

X41 recommends to make use of the USER directive and to install packages using the distribution's package manager whenever possible.

4.2.11 SPBS-CR-22-110: Expired Public Key Cleanup Only on Listing Public Keys

Affected Component:	plugins/auth-backend/src/identity/TokenFactory.ts
Remediation:	NOT

4.2.11.1 Remediation

This issue does not seem to be resolved.

4.2.11.2 Customer Response

> "We do not list keys when generating new ones, so pruning keys during generation would introduce additional latency and/or complexity so this does not seem to be worth the tradeoff. The risk of filling up storage here seems minimal as the keys are very small and generated quite rarely."

4.2.11.3 Description

The auth-backend contains the file *TokenFactory.ts*, which is responsible for generating new tokens. For that purpose, the implementation loads the signing key from a `keyStore`, and utilizes the key to sign a new token. Such keys have a short lifetime, and the *TokenFactory.ts* file rotates the key after a configurable time period.

However, the *TokenFactory.ts* file only cleans expired keys on calls to the function ***listPublicKeys()*** rather than on generating a new signing key.

4.2.11.4 Solution Advice

X41 recommends to clean up expired keys also on the generation of new signing keys.

4.2.12 SPBS-CR-22-111: Missing Operational Security Guidelines

Affected Component:	Documentation
Remediation:	NOT

4.2.12.1 Remediation

Guidelines for maintainers will be updated in the future, currently this issue is not resolved.

4.2.12.2 Customer Response

> "Guidelines for maintainers will be documented in some form, and we'll check in with CNCF and Spotify security to see what they recommend regarding MFA."

4.2.12.3 Description

While certain best practices are already adopted by the project, the posture could still be improved and guidelines created.

The following best practices have already been adopted:

- 2FA[26] is enforced for all members of the Backstage GitHub organization
- Service account credentials are inaccessible to developers
- Pull Requests require approval from at least one other team member
  - Any changes to this setting require re-login with 2FA

However, the only currently used second factors for GitHub are either TOTP[27] or SMS[28], both of which have various downsides and potential for user errors that users should be made aware of.

In the case of TOTP, the TOTP secret could be stored within a password manager along with the password, essentially cancelling out the security gains 2FA with TOTP can provide. Similarly, in the case of SMS-based 2FA, features like iCloud syncing of SMS from an iPhone to a Mac would also cancel out the security gains. In both cases only a single device would need to be compromised to obtain all factors needed for login.

[26] Two Factor Authentication
[27] Time-based One-Time Password
[28] Short Message Service

Additionally, both factors are not resistant to phishing: A convincing phishing website could lead a user to enter both factors on the phishing site, giving the attacker full access.

4.2.12.4 Solution Advice

X41 recommends to develop operational security guidelines for the project, by writing down the already adopted practices and further strengthening security and phishing resistance by heavily encouraging the use of hardware security keys as second factors for day-to-day use. Users should be made aware of the downsides of TOTP and SMS-based 2FA.

In particular, the hardware security keys should support WebAuthn, since WebAuthn challenge responses are bound to domains. Meaning that even if a user enters their password on a phishing site, the phishing site then relays a valid challenge from the legit site to the user and the user signs the challenge with their security key, that challenge will be bound to the domain of the phishing site. This means that the legit site would not accept it and authentication would fail.

While GitHub does require either TOTP or SMS as a second factor before offering the user the option of enrolling a hardware security key, TOTP or SMS should only be used as backup, in case the hardware security key is lost, to make optimal use of the phishing protections hardware security keys with WebAuthn can provide.

4.2.13 SPBS-CR-22-112: Insecure Kubernetes Examples in contrib Folder

Affected Component:	contrib/kubernetes/basic_kubernetes_example_with_helm
Remediation:	NOT

4.2.13.1 Remediation

This issue is not resolved, but additional references will be added to the documentation.

4.2.13.2 Customer Response

> "We'll add appropriate links to the documentation."

4.2.13.3 Description

While reviewing the provided source code it was noticed that the *contrib* folder contains a basic Kubernetes example of how to deploy the Backstage software complex within Kubernetes. Specifically, the files within the folder *contrib/kubernetes/basic_kubernetes_example_with_helm* show how to run the solution in Kubernetes in terms of YAML[29] files.

However, the example lacks some vital security considerations when using Kubernetes, including, but not limited to, the following:

- Missing Role Definitions for RBAC[30]
- Missing Pod Security Policies
- Missing Network Policies
- Missing Resource Policies
- Auto-mount of service account tokens not disabled

4.2.13.4 Solution Advice

X41 recommends to provide some guidance for developers which are new to Kubernetes, especially to provide links to useful documentation regarding best practices for security in Kubernetes.

[29] YAML Ain't Markup Language
[30] Role Based Access Control

4.2.14 SPBS-CR-22-113: Recommended Firewalling of Back-End

Affected Component:	Docs
Remediation:	PARTLY

4.2.14.1 Remediation

The documentation was slightly improved and will get more detailed information about the threat model and a security section in the future.

4.2.14.2 Customer Response

> "Documentation has already been improved somewhat, and this will be documented as part of our upcoming thread model. We'll also add a security section to our deployment documentation, and link to it where needed."

4.2.14.3 Description

In a call with the developers, the testers learned that the back-end should be placed behind an access control system and not exposed directly to untrusted parties such as on the Internet.

This recommendation is not obvious from the Getting Started or related documentation. Moreover, the remark "If the system is not directly accessible over your network the following ports need to be opened"[31] suggests that it is normal for it to be directly accessible from the network. On many server systems, this will include the Internet.

4.2.14.4 Solution Advice

X41 recommends to include the access control recommendation in the setup documentation such that it is not easily overlooked.

[31] `https://backstage.io/docs/getting-started/`

4.2.15 SPBS-CR-22-114: Missing/Broken Authentication for Ancestry Endoint in catalog-backend

Affected Component:	plugins/catalog-backend/src/service/createRouter.ts
Remediation:	FULLY

4.2.15.1 Remediation

The code was modified to pass on the JWT to be verified. This resolves the issue.

```
.get(
'/entities/by-name/:kind/:namespace/:name/ancestry',
async (req, res) => {
  const { kind, namespace, name } = req.params;
  const entityRef = stringifyEntityRef({ kind, namespace, name });
  const response = await entitiesCatalog.entityAncestry(entityRef, {
    authorizationToken: getBearerToken(req.header('authorization')),
  });
  res.status(200).json(response);
},
)
```

Listing 4.21: JWT Handling in Ancestry Endpoint

The corrsponding pull request is 8853[32], with a fix unrelated to the actual issue in pull request 10172[33].

4.2.15.2 Description

The *catalog-backend* provides an endpoint to query for the ancestors of entities within the *createRouter.ts* file. The API endpoint corresponds to a GET endpoint, and does not pass on any authentication token, which is in contrast to other endpoints regarding entities, such as the endpoint *GET /entities/by-name/:kind/:namespace/:name* in the *createRouter.ts* file.

It was verified through inspection of the authentication code that a missing authentication token will lead to a denial of access to the requested resources if the permissions are enabled via the configuration.

[32] https://github.com/backstage/backstage/pull/8853
[33] https://github.com/backstage/backstage/pull/10172

Code Listing 4.22 shows the issue. It is evident that the endpoint does not extract any authentication token from the HTTP request.

```
.get(
    '/entities/by-name/:kind/:namespace/:name/ancestry',
    async (req, res) => {
          const { kind, namespace, name } = req.params;
          const entityRef = stringifyEntityRef({ kind, namespace, name });
          const response = await entitiesCatalog.entityAncestry(entityRef);
          res.status(200).json(response);
    },
)
```

Listing 4.22: Missing Authentication for the Ancestry Endpoint

4.2.15.3 Solution Advice

The code processing the actual request in method ***entitiesCatalog.entityAncestry(entityRef)*** is expecting an authentication token as optional argument and checks if the token is valid.

X41 recommends to fix this functionality defect.

5 About X41 D-Sec GmbH

X41 D-Sec GmbH is an expert provider for application security and penetration testing services. Having extensive industry experience and expertise in the area of information security, a strong core security team of world-class security experts enables X41 D-Sec GmbH to perform premium security services.

X41 has the following references that show their experience in the field:

- Review of the Mozilla Firefox updater[1]
- X41 Browser Security White Paper[2]
- Review of Cryptographic Protocols (Wire)[3]
- Identification of flaws in Fax Machines[4,5]
- Smartcard Stack Fuzzing[6]

The testers at X41 have extensive experience with penetration testing and red teaming exercises in complex environments. This includes enterprise environments with thousands of users and vendor infrastructures such as the Mozilla Firefox Updater (Balrog).

Fields of expertise in the area of application security encompass security-centered code reviews, binary reverse-engineering and vulnerability-discovery. Custom research and IT security consulting, as well as support services, are the core competencies of X41. The team has a strong technical background and performs security reviews of complex and high-profile applications such as Google Chrome and Microsoft Edge web browsers.

X41 D-Sec GmbH can be reached via `https://x41-dsec.de` or `mailto:info@x41-dsec.de`.

[1] `https://blog.mozilla.org/security/2018/10/09/trusting-the-delivery-of-firefox-updates/`
[2] `https://browser-security.x41-dsec.de/X41-Browser-Security-White-Paper.pdf`
[3] `https://www.x41-dsec.de/reports/Kudelski-X41-Wire-Report-phase1-20170208.pdf`
[4] `https://www.x41-dsec.de/lab/blog/fax/`
[5] `https://2018.zeronights.ru/en/reports/zero-fax-given/`
[6] `https://www.x41-dsec.de/lab/blog/smartcards/`

Acronyms

2FA Two Factor Authentication . . . 56
API Application Programming Interface . . . 6
CWE Common Weakness Enumeration . . . 12
DoS Denial of Service . . . 17
HTTP HyperText Transfer Protocol . . . 40
JSON JavaScript Object Notation . . . 39
JWT JSON Web Token . . . 13
LTS Long Term Support . . . 51
PGP Pretty Good Privacy . . . 51
RBAC Role Based Access Control . . . 58
SMS Short Message Service . . . 56
SSRF Server-Side Request Forgery . . . 45
TCP Transmission Control Protocol . . . 39
TOCTOU Time-of-check to time-of-use . . . 23
TOTP Time-based One-Time Password . . . 56
URI Uniform Resource Identifier . . . 40
URL Uniform Resource Locator . . . 21
YAML YAML Ain't Markup Language . . . 58