Source Code Audit of Backstage for the Backstage Team

Final Report and Management Summary

2024-12-16

PUBLIC

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	2024-09-13	Final Report and Management Summary	A. Basma, E. Sesterhenn, JM, M. Vervier, Y. El Baaj
2	2024-12-16	Public Report	M. Vervier

Contents

Dashboard

Target	
Customer	Backstage Team
Name	Backstage
Type	Framework
Version	v1.30.0
Engagement	
Type	White Box Penetration Test
Consultants	5: Ali Basma, Eric Sesterhenn, JM, Markus Vervier, and Yassine El Baaj
Engagement Effort	24 person-days, 2024-08-19 to 2024-09-12
Total issues found	4

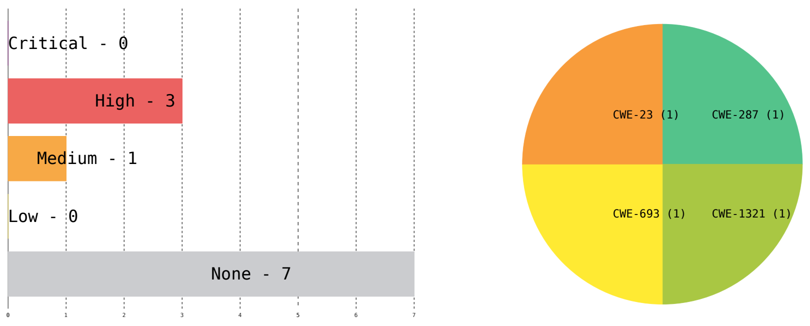


Figure 1: Issue Overview (l: Severity, r: CWE Distribution)

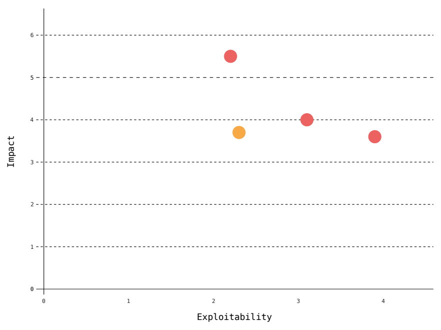


Figure 2: CVSS Impact and Exploitability Distribution

1 Executive Summary

In August and September 2024, X41 D-Sec GmbH performed a security source code audit against Backstage to identify vulnerabilities and weaknesses in the framework, following a previous review in 2023. The work is sponsored by the Open Source Technology Improvement Fund (OSTIF) as part of their ongoing efforts to secure the open source world.

A total of four vulnerabilities were discovered during the test by X41. None were rated as having a critical severity, three as high, one as medium, and none as low. Additionally, seven issues without a direct security impact were identified.

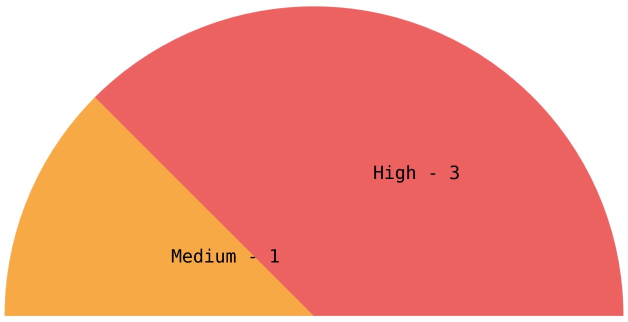


Figure 1.1: Issues and Severity

Backstage is an open-source framework and software catalog that allows developers to quickly ship high-quality code. It is written in TypeScript and extensible via plugins in many ways. As a framework to develop complex applications, vulnerabilities in the platform itself and its core plugins could have a wider scale technical impact on many other systems.

Therefore, X41 was tasked to perform a code audit on the core parts of Backstage. In a source code audit, all information about the system is made available and the code is systematically inspected for security vulnerabilities that are exploitable or even vulnerabilities that are currently shadowed, but have potential to lead to security issues in the future. The test was performed by five experienced security experts between 2024-08-19 and 2024-09-12.

The most severe issue discovered allows an attacker to take over arbitrary accounts in Backstage if multiple authentication providers are enabled. Also a high severity issue allows attackers to perform a Denial of Service attack or, depending on the circumstances, potentially achieve SQL injection or arbitrary code execution. Another issue allows attackers to fetch files outside of the configured TechDocs web root directory from a cloud provider, leading to potential unintended exposure of data. Furthermore, a bypass of the Cross-site Scripting filter was also identified via unfiltered content types.

X41 recommends to implement mitigations against prototype pollution. Possibly dangerous configurations should be marked as such in the documentation, and the Cross-site Scripting filter should be applied based on the output, not on the prediction of an input.

Overall, the Backstage framework appears to be on a good security level compared to source code of similar size and complexity. It is visible that it was designed with security in mind. Nevertheless, due to the complexity and high pace of the development, the potential for vulnerabilities being introduced is always present. It is recommended to perform code audits regularly to detect issues early.

2 Introduction

X41 reviewed Backstage, its 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 a Backstage instance would allow an attacker to gain sensitive information about an organization's digital assets, such as the source code or deployment configurations of software components.

2.1 Methodology

X41 reviewed Backstage as defined in the scope below. The review was mainly based on a source code review alongside analyses of local installations of the software on test systems.

A manual approach for penetration tests and for code reviews is used by X41. This process is supported by tools such as static code analyzers and industry standard web application security tools[1].

X41 adheres to established standards for source code reviewing and penetration testing. These are in particular the *CERT Secure Coding*[2] standards and the *Study - A Penetration Testing Model*[3] of the German Federal Office for Information Security.

In an initial, informal workshop regarding the design and architecture of the application a basic threat model is created. This is used to explore the source code for interesting attack surface and code paths. These are then audited manually and with the help of tools such as static analyzers and fuzzers. The identified issues are documented and can be used in a GAP analysis to highlight changes to previous audits.

[1] `https://portswigger.net/burp`
[2] `https://wiki.sei.cmu.edu/confluence/display/seccode/SEI+CERT+Coding+Standards`
[3] `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	ID	REF
Possible Account Hijacking Via OAuth When Using Built-in Backstage Resolvers	HIGH	BS-CR-24-01	4.1.1
Server-Side Prototype Pollution via Filters Request Parameter	HIGH	BS-CR-24-02	4.1.2
Circumvention of XSS Protection	MEDIUM	BS-CR-24-03	4.1.3
Directory Traversal in TechDocs	HIGH	BS-CR-24-04	4.1.4
Assessing CSRF Risks in GraphQL Content Type Handling	NONE	BS-CR-24-100	4.2.1
Path Disclosure	NONE	BS-CR-24-101	4.2.2
Vulnerable Node.js Modules	NONE	BS-CR-24-102	4.2.3
Express Responses without Explicit Content-Type	NONE	BS-CR-24-103	4.2.4
Local File Path Resolution	NONE	BS-CR-24-104	4.2.5
Lowercasing Emails in AWS Auth	NONE	BS-CR-24-105	4.2.6
Dependency Conflicts during Backstage TechDocs Plugin Installation	NONE	BS-CR-24-106	4.2.7

Table 2.1: Security-Relevant Findings

2.3 Scope

The scope of this audit covered the backstage source code version v1.30.0 as identified by commit `821f065ee9dc781de74c1cfa7ce38b751f1e35b2`[4] from 2024-08-20.

The target of the scope included everything from X41's previous audit of Backstage[5]:

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

In addition to the previous scope, the items listed below were in scope:

- Permissions plugin[6]
- Service to Service Auth[7]
- New backend system[8]
- Public entrypoint[9]

A particular focus was put on the Proxy backend, the authentication service, and other services exported from the `backend-defaults` package[10].

Explicitly out of scope were the items listed below:

- In-depth analysis of the new frontend system, which is in alpha state
- Rate limiting
- Configuration Manager
- Plugin runner backend
- Secrets management

[4] https://github.com/backstage/backstage/tree/821f065ee9dc781de74c1cfa7ce38b751f1e35b2
[5] https://x41-dsec.de/static/reports/X41-Backstage-Audit-2022-Final-Report-PUBLIC.pdf
[6] https://backstage.io/docs/permissions/overview#how-does-it-work
[7] https://backstage.io/docs/auth/service-to-service-auth
[8] https://backstage.io/docs/backend-system/
[9] https://backstage.io/docs/tutorials/enable-public-entry
[10] https://github.com/backstage/backstage/tree/821f065/packages/backend-defaults/src/entrypoints

A Discord chat group was created for direct communication between the testers and the developers, which were available throughout the test and were able to answer any questions on short notice.

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.

X41 set up a local installation of Backstage, using the development version available in the npm[11] repository `https://www.npmjs.com/package/@backstage/create-app` based on the commit in scope as mentioned above.

As a general approach for establishing the use of best practices, the different components and plugins were tested for OWASP[12] Top Ten vulnerabilities[13] and in addition looked for security best practices not covered by OWASP. The source code was audited manually and inspected with the help of semgrep[14] and SonarQube[15]. The project was also searched for violations of its own secure coding practices. Possible discrepancies in URL[16] parsing behavior and related issues such as path and directory traversal were checked.

The following vulnerability classes were investigated in the code that was in-scope:

- Authentication Bypass (OAUTH Misconfiguration, OpenID Misconfiguration)
- Authorization Bypass (Direct-Object-Reference (DOR), Logical Issues)
- Insufficient Secrets Validation (JWT)
- Cryptographic Issues (Signature Bypass, flawed Transport Encryption)
- Information Exposure
- Privilege Escalation
- Prototype Pollution (Server-Side and Client-Side)
- Server-side Injections: stored XSS, SQLi, Command Injection, Template Injection
- Client-side Injections: DOM XSS, mXSS, Template Injection, JavaScript Injection

[11] Node Package Manager
[12] Open Web Application Security Project
[13] `https://owasp.org/www-project-top-ten/`
[14] `https://semgrep.dev/`
[15] `https://www.sonarsource.com/products/sonarqube/`
[16] Uniform Resource Locator

- SSRF[17]
- DoS[18]
- Insecure File Operations (Directory Traversal, Arbitrary File Read/Write, Race-conditions)
- Concurrency Issues (Race-conditions)

The review featured the following items that were investigated during the review:

- Cookies were checked for *Secure* and *HttpOnly* attributes.
- The JWT[19] token validation was checked for potential bypasses or other flaws affecting their validation on the server-side.
- Reliance on Client-Side Authorization and Authentication was investigated.
- The service-to-service authentication was reviewed by inspecting the code and methods used.
- The AWS[20] authentication was investigated for flaws and misconfigurations.
- expressjs responses using ***send()*** were investigated.
- Parsing issues for different formats (JSON, HTTP).
- WebSocket hijacking was investigated using dynamic and static analysis.
- The code was checked for usage of ***url.parse***, ***new URL()***, ***req.url***, ***req.path***, ***req.baseUrl***, ***req.originalUrl***, and URL parsing discrepancies
- The server setup was checked for possible mistakes in TLS[21] configuration and the code was checked for improper trust of client-controlled values, such as *req* attributes.
- The handling of HTTP[22] headers was inspected for the possible handling of untrustworthy input.
- TechDocs import handling was audited on how it sanitizes and handles external data.
- The code and build definitions were inspected for outdated or vulnerable third-party libraries and package using *npm audit* and *OWASP dependency check*.
- Potential Prototype Pollution on the server side was tested statically by inspecting the code manually and with static analysis tools. Injection points were also tested dynamically against pollutions involving *__proto__* and *constructor*.
- Prototype Pollution was tested dynamically on the client side using DOM Invader.
- The code was audited for log-file injections.
- The OAuth authentication using third-party authentication providers, the OpenID layer, the authorization layer as well as the verification of JWT tokens were also audited.
- All *plugins/search-backend-** plugins were tested by first fuzzing the parameters to identify vulnerabilities or error messages that can lead to injection vulnerabilities, and secondly

[17] Server-Side Request Forgery
[18] Denial of Service
[19] JSON Web Token
[20] Amazon Web Services
[21] Transport Layer Security
[22] HyperText Transfer Protocol

by auditing the code.

- DOM[23] Invader was used to to check for DOM-based XSS[24] and other client-side security issues.
- The GraphQL API[25] was also assessed for potential security flaws, including injections and other common vulnerabilities. In particular it was inspected for information exposure.
- All user-controllable inputs were thoroughly tested for potential SSTI[26] vulnerabilities.
- The code was also vetted for potential SSRF attacks, especially in the different URL readers.
- The Backstage Scaffolder plugin was reviewed for different vulnerability classes such as XSS, Template Injection, SQLi, DoS, SSRF.
- File operations were investigated for local file inclusions and directory traversals.
- SQLi via parameters such as filters was investigated.
- The database abstraction layers were investigated for injection attacks.
- The Kubernetes related code was inspected for common flaws.
- Investigated click-jacking via Burp Active Scan.
- Docker files were investigated using the docker vulnerability checker and manual inspection.
- HTTP desync and request smuggling techniques were investigated using Burp.
- Header manipulation and logical issues regarding request headers were investigated using Burp and curl.

2.5 Recommended Further Tests

X41 recommends to perform a retest of the solutions implemented to mitigate the vulnerabilities identified during this review.

To cover more attack surface, a source code auditing of the plugins that were not in scope is also recommended. Additionally, X41 recommends to perform a penetration test of a fully configured instance of Backstage with a more clearly defined threat model as some vulnerabilities might only occur in certain scenarios and configurations.

[23] Document Object Model
[24] Cross-site Scripting
[25] Application Programming Interface
[26] Server-Side Template Injection

Regarding the bug class of Prototype Pollution (CWE 1321), it is recommended to further investigate Backstage for vulnerable code patterns such as the following:

```
myval = object[key] // attacker controls value of `key`
...
myval[key2] = anotherval // attacker controls `key2` and `anotherval`
```

Listing 2.1: r

More information can be found at the *OWASP Prototype Pollution Cheat Sheet*[27].

[27] `https://cheatsheetseries.owasp.org/cheatsheets/Prototype_Pollution_Prevention_Cheat_Sheet.html`

3 Rating Methodology

Security vulnerabilities are given a purely technical rating by the testers when they are discovered during a test. Business factors and financial risks for Backstage Team are beyond the scope of a penetration test, which focuses entirely on technical factors. However, 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.

The CVSS[1] is used to score all findings relevant to security. The resulting CVSS score is mapped to qualitative ratings as shown below.

3.1 CVSS

All findings relevant to security are rated by the testers using the CVSS industry standard version 3.1, revision 1.

Vulnerabilities scored with CVSS get a numeric value based on several metrics ranging from 0.0 (least worst) to 10.0 (worst).

The score captures different factors that express the impact and the ease of exploitation of a vulnerability among other factors. For a detailed description of how the scores are calculated, please see the CVSS version 3.1 specification.[2]

The metrics used to calculate the final score are grouped into three different categories.

[1] Common Vulnerability Scoring System
[2] `https://www.first.org/cvss/v3-1/cvss-v31-specification_r1.pdf`

The *Base Metric Group* represents the intrinsic and fundamental characteristics of a vulnerability that are constant over time and user environments. It captures the following metrics:

- Attack Vector (AV)
- Attack Complexity (AC)
- Privileges Required (PR)
- User Interaction (UI)
- Scope (S)
- Confidentiality Impact (C)
- Integrity Impact (I)
- Availability Impact (A)

The *Temporal Metric Group* represents the characteristics of a vulnerability that change over time but not among user environments. The following metrics are covered by it:

- Exploitability (E)
- Remediation Level (RL)
- Report Confidence (RC)

The *Environmental Metric Group* represents the characteristics of a vulnerability that are relevant and unique to a particular user's environment. It includes the following metrics:

- Attack Vector (MAV)
- Attack Complexity (MAC)
- Privileges Required (MPR)
- User Interaction (MUI)
- Confidentiality Requirement (MCR)
- Integrity Requirement (MIR)
- Availability Requirement (MAR)
- Scope (MS)
- Confidentiality Impact (MC)
- Integrity Impact (MI)
- Availability Impact (MA)

A CVSS vector defines a specific set of metrics and their values, and it can be used to reproduce and assess a given score. It is rendered as a string that exactly reproduces a score.

For example, the vector `CVSS:3.1/AV:N/AC:H/PR:L/UI:R/S:C/C:H/I:L/A:N` defines a base score metric with the following parameters:

- Attack Vector: Network
- Attack Complexity: High
- Privileges Required: Low
- User Interaction: Required
- Scope: Changed
- Confidentiality Impact: High
- Integrity Impact: Low
- Availability Impact: None

In this example, a network-based attacker performs a complex attack after gaining access to some privileges, by tricking a user into performing some actions. This allows the attacker to read confidential data and change some parts of that data.

The detailed scores are the following:

Metric	Score
CVSS Base Score	6.5
Impact Sub-Score	4.7
Exploitability Sub-Score	1.3
CVSS Temporal Score	Not Available
CVSS Environmental Score	Not Available
Modified Impact Sub-Score	Not Available
Overall CVSS Score	6.5

CVSS vectors can be automatically parsed to recreate the score, for example, with the CVSS calculator provided by FIRST, the organization behind CVSS: `https://www.first.org/cvss/calculator/3.1#CVSS:3.1/AV:N/AC:H/PR:L/UI:R/S:C/C:H/I:L/A:N`.

3.2 Severity Mapping

To help in understanding the results of a test, numeric CVSS scores are mapped to qualitative ratings as follows:

Severity Rating	CVSS Score
NONE	0.0
LOW	0.1–3.9
MEDIUM	4.0–6.9
HIGH	7.0–8.9
CRITICAL	9.0–10.0

3.3 Common Weakness Enumeration

The CWE[3] is a set of software weaknesses that allows vulnerabilities and weaknesses in software to be categorized. If applicable, X41 gives a CWE ID for each vulnerability that is discovered during a test.

CWE is a very powerful method for categorizing a vulnerability. It gives general descriptions and solution advice on recurring vulnerability types. CWE is developed by *MITRE*.[4] More information can be found on the CWE site at `https://cwe.mitre.org/`.

[3] Common Weakness Enumeration
[4] `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 BS-CR-24-01: Possible Account Hijacking Via OAuth When Using Built-in Backstage Resolvers

Severity:	HIGH
CVSS v3.1 Total Score:	7.7
CVSS v3.1 Vector:	CVSS:3.1/AV:N/AC:H/PR:N/UI:N/S:U/C:H/I:H/A:L
CWE:	287 – Improper Authentication
Affected Component:	Authentication

4.1.1.1 Description

X41 identified that the use of the built-in resolvers `emailLocalPartMatchingUserEntityName` and `usernameMatchingUserEntityName` can lead to account hijacking. If the victim does not already have an account on the target authentication provider, the use of one of these resolvers could allow an attacker to perform one of the following attacks:

- If the external authentication provider is configured to use the `emailLocalPartMatching-`

`UserEntityName`, the attacker can create an account with a specially crafted email address that matches the username of the victim and hijack their account.

- If the external authentication provider is configured to use the `usernameMatchingUserEntityName`, the attacker can create an account with the same username as of the victims and hijack their account.

Backstage can be configured to have any number of OAuth external authentication providers such as GitHub or Google[1]. Once a user enters the correct credentials for the external authentication provider, Backstage will check if that user can be mapped to a Backstage user identity, which typically lives in the Backstage Catalog. The condition that decides whether a user authenticated through an external authentication provider indeed exists in the Backstage Catalog is determined by a Backstage Resolver.

In addition to that, the official documentation mentions that the use of the `usernameMatchingUserEntityName` is specific to GitHub[2]. However, the reviewed version of Backstage allows using it with other authentication providers such as GitLab. This matches the documentation at `https://backstage.io/docs/auth/gitlab/provider#configuration`.

This means that a victim that is not registered on GitLab, using GitHub as an authentication provider and registered there with the username *`johndoe`* can see their account hijacked by an attacker if the latter creates a new GitLab account with the username *`johndoe`* or an email address with a local part matching that same username, namely *`johndoe@attacker-controlled.com`*.

4.1.1.2 Solution Advice

X41 recommends to set a priority order between the different resolvers in the documentation and recommend the use of the `emailMatchingUserEntityProfileEmail` resolver, which performs the user resolution using the full email address. This means that an attacker cannot take over an account using the methods described above, unless they manage to obtain access to the victim's email address. Additionally, X41 recommends to explicitly label those resolvers as being dangerous as they are subject to attacker's abuse.

[1] `https://backstage.io/docs/getting-started/config/authentication`
[2] `https://backstage.io/docs/auth/identity-resolver/#using-builtin-resolvers`

4.1.2 BS-CR-24-02: Server-Side Prototype Pollution via Filters Request Parameter

Severity:	HIGH
CVSS v3.1 Total Score:	7.5
CVSS v3.1 Vector:	CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:N/I:N/A:H
CWE:	1321 – Improperly Controlled Modification of Object Prototype Attributes ('Prototype Pollution')
Affected Component:	packages/core-components/src/components/Table/Table.tsx:extractValueByField()

4.1.2.1 Description

X41 discovered that the server-side code responsible for parsing and processing the *filter* parameter is subject to a prototype pollution vulnerability[3].

By polluting the prototype of an object, attackers are able to cause unintended behavior in the server-side NodeJS code, leading to a denial-of-service condition or even the injection and execution of unintended database queries. Depending on the context and conditions, prototype pollution can even lead to arbitrary code execution.

The filter parameter contains a value key called *__proto__*, as seen in the request in listing 4.1.

```
GET /api/catalog/entities/by-query?limit=0&filter=foo=guest,__proto__=`"' HTTP/1.1
Host: localhost:7007
authorization: Bearer eyJ0eXAiOiJ2bmQuYmFja3N0YW...
Content-Type: application/json
Accept: */*
Origin: http://localhost:3000
Sec-Fetch-Site: same-site
Sec-Fetch-Mode: cors
Sec-Fetch-Dest: empty
Referer: http://localhost:3000/
Connection: keep-alive
```

Listing 4.1: Polluting HTTP Request with Filter Parameter

This will result in an HTTP error 500 being returned by the server and cause errors for all subsequent calls to ***Database.prepare()*** as shown in listing 4.2.

[3] https://portswigger.net/research/server-side-prototype-pollution#what-is-prototype-pollution

```
[1] 2024-08-28T19:55:16.506Z rootHttpRouter error Request failed with status 500 select * from `user_info` where `user_entity_ref` = 'user:development/guest' and `values` = '`"''' limit 1 - no such column: values type=errorHandler code=SQLITE_ERROR stack=SqliteError: select * from `user_info` where `user_entity_ref` = 'user:development/guest' and `values` = '`"''' limit 1 - no such column: values
[1]     at Database.prepare (/data/backstage/node_modules/better-sqlite3/lib/methods/wrappers.js:5:21)
[SNIP]
[1]     at UserInfoDatabaseHandler.getUserInfo (/data/backstage/node_modules/@backstage/plugin-auth-backend/src/identity/UserInfoDatabaseHandler.ts:51:18)
[1]     at <anonymous> (/data/backstage/node_modules/@backstage/plugin-auth-backend/src/identity/router.ts:102:22)
[1] 2024-08-28T19:55:17.630Z catalog warn Failed to load processing items update `refresh_state` set `next_update_at` = datetime('now', '138.6517332388715 seconds'), `values` = '`"''' where 1 = 0 - no such column: values code=SQLITE_ERROR stack=SqliteError: update `refresh_state` set `next_update_at` = datetime('now', '138.6517332388715 seconds'), `values` = '`"''' where 1 = 0 - no such column: values
[1]     at Database.prepare (/data/backstage/node_modules/better-sqlite3/lib/methods/wrappers.js:5:21)
[SNIP]
[1]     at DefaultProcessingDatabase.getProcessableEntities (/data/backstage/node_modules/@backstage/plugin-catalog-backend/src/database/DefaultProcessingDatabase.ts:234:11)
[1]     at options.database.transaction.doNotRejectOnRollback (/data/backstage/node_modules/@backstage/plugin-catalog-backend/src/database/DefaultProcessingDatabase.ts:289:20)
```

Listing 4.2: Database Error Due to Polluted Object Prototype

As seen in the log, the value `"' became part of a query to the SQL[4] database in escaped form, but caused an error due to the query being incorrectly formed. The reason for this lies in the previous request and (seemingly) unrelated code found in file *packages/core-components/src/components/Table/Table.tsx* and shown in listing 4.3.

[4] Structured Query Language

```
function extractValueByField(data: any, field: string): any | undefined {
  const path = field.split('.');
  let value = data[path[0]];

  for (let i = 1; i < path.length; ++i) {
    if (value === undefined) {
      return value;
    }

    const f = path[i]; // MARK1 const f is extract from the untrustworthy path input and assumed
    ↪  to be '__proto__'
    value = value[f]; // MARK2 value[f] is setting value to it`s own prototype property like if
    ↪  it was value['__proto__']
  }

  return value; // prototype of 'value' is returned instead of the object 'value'
}
```

Listing 4.3: Code Polluting the Database Value Object's Prototype

Since the function returns the prototype of the object, modifications of its properties will translate to all other objects that are later derived from its prototype. This seems to be the case and the reason that the value for key *__proto__* in the filters is appearing again in subsequent database requests, breaking the functionality of backstage until the process is restarted. Setting a key to the value *constructor* also causes a pollution situation because a function type is overwritten by a string type.

During the test, no way was identified to achieve direct code execution since the prototype was overwritten by a string type value and not by another object or array value. This prevents classical exploitation or access to prototype pollution gadgets[5]. Also the SQL query could not be subverted maliciously since the polluted value was still correctly escaped by the prepared statement handling code. However, it cannot be ruled out that with more time available, an accessible gadget could be found, or that a database operation that does not escape the value properly could be triggered.

4.1.2.2 Solution Advice

It is strongly recommended to not use untrustworthy external input directly as key values on objects and arrays in the TypeScript and in JavaScript code. If they are used as keys, it is strongly

[5] https://github.com/BlackFan/client-side-prototype-pollution

recommended to explore different methods to prevent a pollution of the prototype, for example setting hardening flags on NodeJS that prevent the use of the $\_\_proto\_\_$ key, or by freezing of object prototypes that will prevent them from being modified. More information can be found in the *OWASP Prototype Pollution Cheat Sheet*[6].

[6] `https://cheatsheetseries.owasp.org/cheatsheets/Prototype_Pollution_Prevention_Cheat_Sheet.html`

4.1.3 BS-CR-24-03: Circumvention of XSS Protection

Severity:	MEDIUM
CVSS v3.1 Total Score:	6.5
CVSS v3.1 Vector:	CVSS:3.1/AV:N/AC:L/PR:L/UI:R/S:C/C:L/I:L/A:L
CWE:	693 – Protection Mechanism Failure
Affected Component:	plugins/techdocs-node/src/stages/publish/helpers.ts:getContent-TypeForExtension()

4.1.3.1 Description

The function ***getHeadersForFileExtension()*** is used to set the *`Content-Type`* of files served via TechDocs. It calls the helper function ***getContentTypeForExtension()*** to resolve the MIME[7] type based on the file extension. To prevent XSS, a filter is applied to not serve HTML[8], XML[9] and SVG[10] files, as shown in listing 4.4. This filter is applied by various TechDocs file storage providers such as *`awsS3`* and *`GoogleStorage`*, since the storage might be controlled by external parties.

```
const getContentTypeForExtension = (ext: string): string => {
  const defaultContentType = 'text/plain; charset=utf-8';

  // Prevent sanitization bypass by preventing browsers from directly rendering
  // the contents of untrusted files.
  if (ext.match(/htm|xml|svg/i)) {
    return defaultContentType;
  }

  return mime.contentType(ext) || defaultContentType;
};
```

Listing 4.4: MIME-Type Filtering

Several different MIME types exist that allow the execution of JavaScript[11]. By cross-referencing these with the mime-db[12] used by Backstage, the following unfiltered file extensions could be identified:

[7] Multipurpose Internet Mail Extensions
[8] HyperText Markup Language
[9] Extensible Markup Language
[10] Scalable Vector Graphics
[11] `https://github.com/BlackFan/content-type-research/blob/master/XSS.md`
[12] `https://github.com/jshttp/mime-db/blob/master/db.json`

- *.appcache* (`text/cache-manifest`)
- *.manifest* (`text/cache-manifest`)
- *.mathml* (`application/mathml+xml`)
- *.owl* (`application/octet-stream`)
- *.rdf* (`application/rdf+xml`)
- *.rng* (`application/xml`)
- *.vtt* (`text/vtt`)
- *.xht* (`application/xhtml+xml`)
- *.xsd* (`application/xml`)
- *.xsl* (`application/xml`)

Depending on the browser and final CSP[13] set, these can allow the execution of JavaScript in the victim's browser.

One example that executes JavaScript is shown in listings 4.5 and 4.6.

```
<a:script xmlns:a="http://www.w3.org/1999/xhtml" src="hi.js" type="application/javascript"/>
```

Listing 4.5: XHT File Loading JavaScript

```
alert(1337);
```

Listing 4.6: JavaScript File hi.js

4.1.3.2 Solution Advice

X41 recommends to not only exclude file extensions before the MIME lookup, but filtering MIME types after the lookup as well. Another option to mitigate this issue is to adopt an allowlist-based filter that only allows known-safe content to be rendered.

Additionally, it should be investigated whether the serving of TechDocs can be further restricted by using a less permissive CSP header, or only rendering the file contents in a sandboxed iframe[14].

[13] Content Security Policy
[14] `https://developer.mozilla.org/en-US/docs/Web/HTML/Element/iframe#sandbox`

4.1.4 BS-CR-24-04: Directory Traversal in TechDocs

Severity:	HIGH
CVSS v3.1 Total Score:	7.7
CVSS v3.1 Vector:	CVSS:3.1/AV:N/AC:L/PR:L/UI:N/S:C/C:H/I:N/A:N
CWE:	23 – Relative Path Traversal
Affected Component:	plugins/techdocs-node/src/stages/publish/awsS3.ts

4.1.4.1 Description

When using the *awsS3* storage provider for TechDocs with a *bucketRootPath*, it is possible to access files outside the configured *bucketRootPath*.

The function ***docsRouter()*** effectively passes the user-controlled *req.path* to ***path.posix.join()***, which joins the paths *and then normalizes them*[15], as shown in listing 4.7.

```
docsRouter(): express.Handler {
    return async (req, res) => {
        const decodedUri = decodeURI(req.path.replace(/^\//, ''));

        // filePath example - /default/component/documented-component/index.html
        const filePathNoRoot = this.legacyPathCasing
        ? decodedUri
        : lowerCaseEntityTripletInStoragePath(decodedUri);

        // Prepend the root path to the relative file path
        const filePath = path.posix.join(this.bucketRootPath, filePathNoRoot);
        // [SNIP]
        const resp = await this.storageClient.send(
            new GetObjectCommand({ Bucket: this.bucketName, Key: filePath }),
        );
        // [SNIP]
        res.send(await streamToBuffer(resp.Body as Readable));
    } // [SNIP]
};
```

Listing 4.7: Request Handling of S3 Provider

By sending a specially crafted HTTP path, it is possible to perform directory traversal and access objects outside the *bucketRootPath*.

[15] https://nodejs.org/api/path.html#pathjoinpaths

Although *bucketRootPath* was configured to point to the /static/ directory, X41 managed to access files in the /secret/ directory using the command shown in listing 4.8.

```
curl -H "Cookie: $BACKSTAGE_AUTH" --path-as-is 'http://localhost:7007/api/techdocs/static/docs/de
↪ fault/component/backstage/index.html/../../../../../secret/secret.txt'
```

Listing 4.8: Path Traversal Payload on awsS3 Using curl Command

This returned the contents of the /secret/secret.txt file within the bucket.

GoogleStorage handles requests in a very similar way and is likely also affected. *OpenStackSwift* and *AzureBlobStorage* do not support a *bucketRootPath*. The *Local* TechDocs provider uses ***express.static()*** and X41 believes that it is not affected.

4.1.4.2 Solution Advice

X41 recommends to make use of ***resolveSafeChildPath()***, as documented in the project's *SECURITY.md*[16].

[16] https://github.com/backstage/backstage/blob/821f065/SECURITY.md#local-file-path-resolution

4.2 Informational Notes

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 BS-CR-24-100: Assessing CSRF Risks in GraphQL Content Type Handling

Affected Component:	https://demo. backstage.io/api/graphql

4.2.1.1 Description

X41 identified that while the GraphQL API primarily accepts *application/json* as a content type, which is generally considered secure against CSRF[17] attacks due to the same-origin policy enforced by modern browsers, it also supports the *application/x-www-form-urlencoded* content type. A GraphQL endpoint accepting form content types is often a first step to achieve CSRF attacks[18].

When an API allows users to update their email address, a mutation request might look like the request shown in listing 4.9.

```
POST /update-profile HTTP/1.1
Host: api.example.com
Content-Type: application/json

{
  "email": "newemail@example.com"
}
```

Listing 4.9: Request with application/json Content Type

If this request is sent cross-origin with the *application/json* content type, the browser will first send an HTTP *OPTIONS* preflight request. The server must respond with the appropriate CORS[19]

[17] Cross-Site Request Forgery
[18] https://medusa0xf.medium.com/how-to-perform-csrf-attack-in-graphql-api-fda9594dbd42
[19] Cross-Origin Resource Sharing

headers to permit the actual request. If that same request uses the *application/x-www-form-urlencoded* content type.

```
POST /update-profile HTTP/1.1
Host: api.example.com
Content-Type: application/x-www-form-urlencoded

email=newemail%40example.com
```

Listing 4.10: Request with application/x-www-form-urlencoded Content Type

The browser would send it directly without a preflight, as this is a "simple" request. Stricter checks for *application/json* help prevent attacks like CSRF, where malicious sites might exploit authenticated users to perform unintended actions on other sites.

When changing the content type to *application/x-www-form-urlencoded*, X41 was unable to fully exploit this vulnerability because there were no exposed mutations in the GraphQL schema. The presence of a mutation in the future could enable an attacker to exploit this vulnerability to execute unauthorized actions on behalf of authenticated users.

4.2.1.2 Solution Advice

X41 recommends enforcing strict content type validation by configuring the GraphQL API to only accept *application/json* requests. This restriction will prevent the API from accepting potentially exploitable content types like *application/x-www-form-urlencoded*.

4.2.2 BS-CR-24-101: Path Disclosure

Affected Component:	Multiple

4.2.2.1 Description

When errors such as authentication failures are thrown in the code, the exception is presented to the users. This leaks the full path of the backstage installation, which can help attackers with further attacks. A request with an invalid, but correctly formatted JWT (see listing 4.11) will result in the error shown in listing 4.12.

```
GET /api/catalog/.well-known/backstage/permissions/apply-conditions HTTP/1.1
Host: localhost:7007
authorization: Bearer eyJ0i....
```

Listing 4.11: Request with Bad JWT

```
{
   "error" : {
      "cause" : {
         "code" : "ERR_JWKS_NO_MATCHING_KEY",
         "message" : "no applicable key found in the JSON Web Key Set",
         "name" : "JWKSNoMatchingKey",
         "stack" : "JWKSNoMatchingKey: no applicable key found in the JSON Web Key Set\n
        at LocalJWKSet.getKey (/app/node_modules/jose/dist/node/cjs/jwks/local.js:84:19)\n
        at RemoteJWKSet.localJWKSet [as _local]
        ↪  (/app/node_modules/jose/dist/node/cjs/jwks/local.js:115:63)\n
        at RemoteJWKSet.getKey (/app/node_modules/jose/dist/node/cjs/jwks/remote.js:79:31)\n
        at remoteJWKSet (/app/node_modules/jose/dist/node/cjs/jwks/remote.js:119:64)\n
        at flattenedVerify (/app/node_modules/jose/dist/node/cjs/jws/flattened/verify.js:75:21)\n
        at compactVerify (/app/node_modules/jose/dist/node/cjs/jws/compact/verify.js:18:60)\n
        at Object.jwtVerify (/app/node_modules/jose/dist/node/cjs/jwt/verify.js:8:58)\n
        at UserTokenHandler.verifyToken
        ↪  (/app/node_modules/@backstage/backend-defaults/dist/auth.cjs.js:892:36)\n
        at process.processTicksAndRejections (node:internal/process/task_queues:95:5)\n
        at async DefaultAuthService.authenticate
        ↪  (/app/node_modules/@backstage/backend-defaults/dist/auth.cjs.js:44:24)\n
        at async #extractCredentialsFromRequest
        ↪  (/app/node_modules/@backstage/backend-defaults/dist/httpAuth.cjs.js:50:12)\n
        at async DefaultHttpAuthService.credentials
        ↪  (/app/node_modules/@backstage/backend-defaults/dist/httpAuth.cjs.js:74:96)"
      },
        "message" : "Invalid token; caused by JWKSNoMatchingKey: no applicable key found in the
        ↪  JSON Web Key Set",
        "name" : "AuthenticationError"
    },
    "request" : {
        "method" : "GET",
        "url" : "/api/catalog/.well-known/backstage/permissions/apply-conditions"
    },
    "response" : {
        "statusCode" : 401
    }
}
```

Listing 4.12: Response Leaking Path

4.2.2.2 Solution Advice

X41 suggests to hide the full stack traces from users, log them to a file and show a generic error message instead.

4.2.3 BS-CR-24-102: Vulnerable Node.js Modules

Affected Component:	npm Dependencies

4.2.3.1 Description

Using the `yarn npm audit -all`, X41 identified that the following packages were vulnerable to publicly disclosed vulnerabilities:

- The `ws` package version *8.14.2* is vulnerable to DoS when handling a request with many HTTP headers. Additionally, a publicly available exploit can be found in the security advisory under: `https://github.com/advisories/GHSA-3h5v-q93c-6h6q`.
- The `Webpack` package version *5.91.0* is vulnerable to DOM clobbering that leads to XSS. A publicly available exploit can be found in the security advisory under: `https://github.com/advisories/GHSA-4vvj-4cpr-p986`.

X41 noticed that the `ws` package was upgraded for the main application, as visible under the GitHub Pull Request 25699[20]. However, this was not applied for the `signals-node` plugin.

Since no concrete way to exploit both vulnerabilities was found, this is reported as an informational note.

4.2.3.2 Solution Advice

X41 recommends to upgrade the vulnerable packages to the patched versions mentioned in their respective security advisories. Additionally, X41 recommends to add the path to each manifest file to Dependabot[21] for a protection spanning all the application's components.

[20] `https://github.com/backstage/backstage/pull/25699`
[21] `https://github.com/dependabot`

4.2.4 BS-CR-24-103: Express Responses without Explicit Content-Type

Affected Component:	Source Code

4.2.4.1 Description

The Backstage project's *SECURITY.md* mandates[22] that the Express responses' ***.send(...)*** function should not be used, unless a non-JSON[23] content type is sent.

X41 found several violations, such as the one shown in listing 4.13[24].

```
router.get('/status', async (req: Request<any, NotificationStatus>, res) => {
  const user = await getUser(req);
  const status = await store.getStatus({ user });
  res.send(status);
});
```

Listing 4.13: Use of res.send()

The violations found in `plugins/notifications-backend/src/service/router.ts` are listed below.

- Line 240-243[25]
- Line 262[26]
- Line 265[27]
- Line 304 [28]

X41 used a manual approach to identify the above list of violations, which may not be exhaustive.

[22] `https://github.com/backstage/backstage/blob/821f065/SECURITY.md#express-responses`
[23] JavaScript Object Notation
[24] `https://github.com/backstage/backstage/blob/821f065/plugins/notifications-backend/src/service/router.ts#L249`
[25] `https://github.com/backstage/backstage/blob/821f065/plugins/notifications-backend/src/service/router.ts#L240-L243`
[26] `https://github.com/backstage/backstage/blob/821f065/plugins/notifications-backend/src/service/router.ts#L262`
[27] `https://github.com/backstage/backstage/blob/821f065/plugins/notifications-backend/src/service/router.ts#L265`
[28] `https://github.com/backstage/backstage/blob/821f065/plugins/notifications-backend/src/service/router.ts#L304`

4.2.4.2 Solution Advice

X41 recommends to implement an automation (e.g. using semgrep[29]) that detects and rejects source code violating the project's coding practices.

[29] `https://github.com/semgrep/semgrep`

4.2.5 BS-CR-24-104: Local File Path Resolution

Affected Component:	/docs/features/software-templates/writing-custom-actions.md

4.2.5.1 Description

The Backstage project's *SECURITY.md* mandates[30] the use of ***resolveSafeChildPath()*** when accessing local file paths to protect against directory traversals.

X41 found at least one violation[31] of this rule in example source code templates, as shown in listing 4.14.

```
async handler(ctx) {
   const { signal } = ctx;
   await writeFile(
      `${ctx.workspacePath}/${ctx.input.filename}`,
      ctx.input.contents,
      { signal },
      _ => {},
   );
},
```

Listing 4.14: Path Joined in an Unsafe Manner

4.2.5.2 Solution Advice

X41 recommends to implement an automation (e.g. using semgrep[32]) that detects and rejects source code violating the project's coding practices.

[30] https://github.com/backstage/backstage/blob/821f065/SECURITY.md#local-file-path-resolution
[31] https://github.com/backstage/backstage/blob/821f065/docs/features/software-templates/writing-custom-actions.md?plain=1#L121
[32] https://github.com/semgrep/semgrep

4.2.6 BS-CR-24-105: Lowercasing Emails in AWS Auth

Affected Component:	plugins/auth-backend-module-aws-alb-provider/src/authenticator.ts

4.2.6.1 Description

X41 identified a potential issue related to how email addresses are handled in the AWS authentication code. The application transforms all email addresses to lowercase for username creation, as shown in listing 4.15. This could lead to a vulnerability if the email server is case sensitive.

```
const fullProfile: PassportProfile = {
    provider: 'unknown',
    id: claims.sub,
    displayName: claims.name,
    username: claims.email.split('@')[0].toLowerCase(),
    name: {
      familyName: claims.family_name,
      givenName: claims.given_name,
    },
    emails: [{ value: claims.email.toLowerCase() }],
    photos: [{ value: claims.picture }],
  };
```

Listing 4.15: AWS Authentication Code

While most major email providers (such as Gmail, Outlook, and Yahoo) treat email addresses as case insensitive, the RFC[33] 5321 standard allows for case sensitivity in the local part of email addresses. If an organization uses a custom email server configured to be case sensitive, this could lead to security issues.

If two users register with the same email but with different cases (e.g., User@example.com and user@example.com), they would be considered the same user due to the lowercasing. This could lead to user account overlap, unauthorized access, and other security risks, especially in environments using case-sensitive email systems.

4.2.6.2 Solution Advice

X41 recommends to consider how email addresses are handled in AWS authentication and to automatically lowercase the local part of email addresses should be avoided when generating

[33] Request for Comments

usernames, as this could cause conflicts if the email server treats email addresses as case sensitive. Instead, ensure that email addresses are stored and compared exactly as provided by the user, or enforce consistent case-insensitivity throughout the system.

4.2.7 BS-CR-24-106: Dependency Conflicts during Backstage TechDocs Plugin Installation

Affected Component:	@backstage/plugin-techdocs

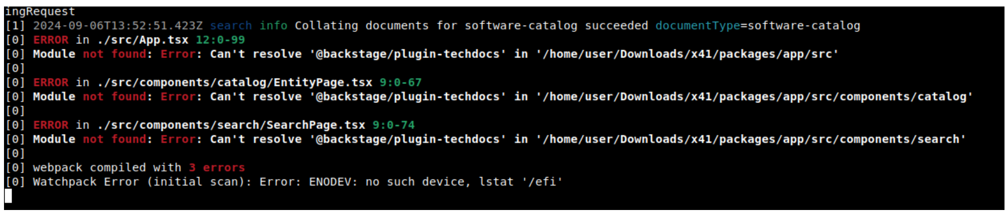
4.2.7.1 Description

During the installation of the Backstage TechDocs plugin, a significant dependency conflict was identified between the frontend and backend components. Specifically, the installation process caused one component's files to overwrite or remove critical files from the other component, leading to a broken setup.

Installing the frontend plugin `backstage/plugin-techdocs` using the command-line shown in listing 4.16 adds the necessary files, including the `plugin-techdocs-backend` directory within `node_modules/backstage/`. However, this installation also removes the `plugin-techdocs` frontend plugin from the `node_modules/backstage/`. This will provoke an error message, as shown in figure 4.1.

```
yarn --cwd packages/app add @backstage/plugin-techdocs
```

Listing 4.16: TechDocs Frontend Installation



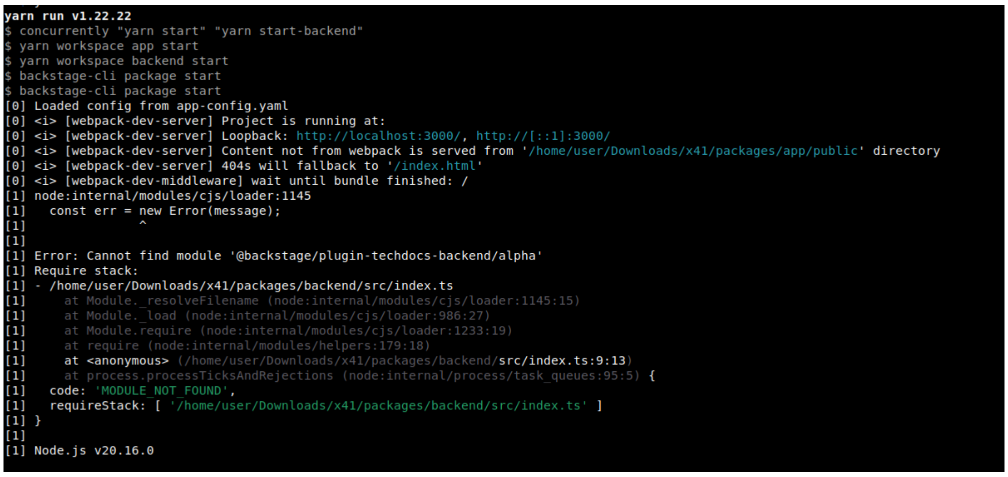
```
ingRequest
[1] 2024-09-06T13:52:51.423Z search info Collating documents for software-catalog succeeded documentType=software-catalog
[0] ERROR in ./src/App.tsx 12:0-99
[0] Module not found: Error: Can't resolve '@backstage/plugin-techdocs' in '/home/user/Downloads/x41/packages/app/src'
[0]
[0] ERROR in ./src/components/catalog/EntityPage.tsx 9:0-67
[0] Module not found: Error: Can't resolve '@backstage/plugin-techdocs' in '/home/user/Downloads/x41/packages/app/src/components/catalog'
[0]
[0] ERROR in ./src/components/search/SearchPage.tsx 9:0-74
[0] Module not found: Error: Can't resolve '@backstage/plugin-techdocs' in '/home/user/Downloads/x41/packages/app/src/components/search'
[0]
[0] webpack compiled with 3 errors
[0] Watchpack Error (initial scan): Error: ENODEV: no such device, lstat '/efi'
```

Figure 4.1: TechDocs Frontend Error Message

Installing the backend plugin should ideally work alongside the frontend. However, when the installation command shown in listing 4.17 is executed, the `plugin-techdocs-backend` directory is removed from `node_modules/backstage/`, leaving the backend incomplete and potentially causing errors. Additionally, the `plugin-techdocs` frontend plugin is re-added. This will provoke the error message shown in figure 4.2 leading to a scenario where you can never have both the frontend and backend plugins installed simultaneously.

```
yarn --cwd packages/backend add @backstage/plugin-techdocs-backend
```

Listing 4.17: TechDocs Frontend Installation



```
yarn run v1.22.22
$ concurrently "yarn start" "yarn start-backend"
$ yarn workspace app start
$ yarn workspace backend start
$ backstage-cli package start
$ backstage-cli package start
[0] Loaded config from app-config.yaml
[0] <i> [webpack-dev-server] Project is running at:
[0] <i> [webpack-dev-server] Loopback: http://localhost:3000/, http://[::1]:3000/
[0] <i> [webpack-dev-server] Content not from webpack is served from '/home/user/Downloads/x41/packages/app/public' directory
[0] <i> [webpack-dev-server] 404s will fallback to '/index.html'
[0] <i> [webpack-dev-middleware] wait until bundle finished: /
[1] node:internal/modules/cjs/loader:1145
[1]   const err = new Error(message);
[1]               ^
[1]
[1] Error: Cannot find module '@backstage/plugin-techdocs-backend/alpha'
[1] Require stack:
[1] - /home/user/Downloads/x41/packages/backend/src/index.ts
[1]     at Module._resolveFilename (node:internal/modules/cjs/loader:1145:15)
[1]     at Module._load (node:internal/modules/cjs/loader:986:27)
[1]     at Module.require (node:internal/modules/cjs/loader:1233:19)
[1]     at require (node:internal/modules/helpers:179:18)
[1]     at <anonymous> (/home/user/Downloads/x41/packages/backend/src/index.ts:9:13)
[1]     at process.processTicksAndRejections (node:internal/process/task_queues:95:5) {
[1]   code: 'MODULE_NOT_FOUND',
[1]   requireStack: [ '/home/user/Downloads/x41/packages/backend/src/index.ts' ]
[1] }
[1]
[1] Node.js v20.16.0
```

Figure 4.2: TechDocs Backend Error Message

4.2.7.2 Solution Advice

To avoid issues with conflicting plugin installations in Backstage TechDocs, it is recommended to review and streamline the installation process for the frontend and backend plugins. Ensuring that both plugins can coexist without removing each other's necessary files would prevent manual intervention.

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:

- Source code audit of ISC BIND9 DNS server[1]
- Source code audit of the Git source code version control system[2]
- Review of the Mozilla Firefox updater[3]
- X41 Browser Security White Paper[4]
- Review of Cryptographic Protocols (Wire)[5]
- Identification of flaws in Fax Machines[6,7]
- Smartcard Stack Fuzzing[8]

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://x41-dsec.de/news/security/research/source-code-audit/2024/02/13/bind9-security-audit/`
[2] `https://x41-dsec.de/security/research/news/2023/01/17/git-security-audit-ostif/`
[3] `https://blog.mozilla.org/security/2018/10/09/trusting-the-delivery-of-firefox-updates/`
[4] `https://browser-security.x41-dsec.de/X41-Browser-Security-White-Paper.pdf`
[5] `https://www.x41-dsec.de/reports/Kudelski-X41-Wire-Report-phase1-20170208.pdf`
[6] `https://www.x41-dsec.de/lab/blog/fax/`
[7] `https://2018.zeronights.ru/en/reports/zero-fax-given/`
[8] `https://www.x41-dsec.de/lab/blog/smartcards/`

Acronyms

API	Application Programming Interface	11
AWS	Amazon Web Services	10
CORS	Cross-Origin Resource Sharing	27
CSP	Content Security Policy	24
CSRF	Cross-Site Request Forgery	27
CVSS	Common Vulnerability Scoring System	13
CWE	Common Weakness Enumeration	16
DOM	Document Object Model	11
DoS	Denial of Service	10
HTML	HyperText Markup Language	23
HTTP	HyperText Transfer Protocol	10
JSON	JavaScript Object Notation	32
JWT	JSON Web Token	10
MIME	Multipurpose Internet Mail Extensions	23
npm	Node Package Manager	9
OWASP	Open Web Application Security Project	9
RFC	Request for Comments	35
SQL	Structured Query Language	20
SSRF	Server-Side Request Forgery	10
SSTI	Server-Side Template Injection	11
SVG	Scalable Vector Graphics	23
TLS	Transport Layer Security	10
URL	Uniform Resource Locator	9
XML	Extensible Markup Language	23
XSS	Cross-site Scripting	11