

# Pentesting Report

## *Data WareHouse (DWH)*

Johannes Merkert á Sebastian Franz á Jan Girlich @ iteratec GmbH

Date: 10.06.2024 - Client: Example Ltd. - Report number: 24-4

**iteratec**

# Table of Contents

1. Summary .....	1
2. Introduction .....	3
3. Organizational information .....	4
4. Scope of the pentest .....	5
4.1. Limitations .....	5
5. Identified issues .....	6
5.1. Finding 24-4:1: Authenticated Remote Code Execution in ETL .....	6
5.2. Finding 24-4:2: ETL Page Contains Execution Logs .....	11
5.3. Finding 24-4:3: Unicorn Application Server Runs as Root Process .....	13
5.4. Finding 24-4:4: Docker Container User is Root .....	15
5.5. Finding 24-4:5: Docker Socket is Mounted in Container .....	17
5.6. Finding 24-4:6: Secrets in Environment File .....	22
5.7. Finding 24-4:7: Linked Databases with Financial Data in MSSQL .....	25
5.8. Finding 24-4:8: Authenticated Remote Code Execution in MSSQL .....	29
5.9. Finding 24-4:9: Insecure Authorization from JWT .....	33
5.10. Finding 24-4:10: Incorrect Handling of Preflight Request Leads to OAuth Redirect .....	39
5.11. Finding 24-4:11: Pages in Admin Section Accessible for Non-Admin Users .....	42
5.12. Befund 24-4:12: Credentials in Git repository .....	44
5.13. Finding 24-4:13: Vulnerable Dependencies .....	47
5.14. Finding 24-4:14: Shared User Account on Host .....	49
5.15. Finding 24-4:15: Passwordless Sudo .....	51
6. Appendix .....	53
6.1. EXAMPLE: Results npm audit - TEST .....	53



# 1. Summary

The issues found during this test can be grouped in four categories, each relating to a certain way of attacks.

Attack chain for a full system compromise through the Admin web interface

1. The first attack chain starts with a Remote Code Execution in the ETL. Arbitrary commands can be injected in the POST request for defining the ETL steps to execute. (24-4:1) It was possible to see the result of injected commands in the ETL logs, which simplified this attack. (24-4:2)
2. The injected code runs in the context of the Unicorn app server, which runs with root privileges. (24-4:3) This allows to take full control of the Docker container in which Unicorn is running.
3. This Docker container's default user is root as well. (24-4:4) Additionally, the Docker container has the Docker API mounted.
4. With the Docker API and root privileges it is possible to break out of the Docker container and escalate to root on the host, leading to a full system compromise. (24-4:5)

Lateral movement to database servers including remote code execution

From the compromised host machine in the above attack chain, it is possible to move laterally to other systems.

1. Key to this further attack chain is an environment file found on the host machine. It includes plain-text secrets to access other servers. (24-4:6)
2. One of the secrets are the credentials of the connected MSSQL database which is also linked to another database containing financial from 2020 up until the time of testing. (24-4:7)
3. The MSSQL database has also the feature xp\_cmdshell activated, which allows for Remote Code Execution on the host. The privileges the database process can be elevated to NT AUTHORITY\SYSTEM with publicly available tools. (24-4:8)

Attacks on the authorization

The above attack chain requires the Admin role, which can be acquired in certain situations.

- ¥ If an attacker gains control of the traefik proxy in front of the DWH application, they can gain Admin privileges in the web interface. (24-4:9)
- ¥ The server also incorrectly handles a preflight request. Although the observed behavior

is not in accordance with the OAuth standard and should be fixed, it does not pose an immediate security threat. (24-4:10)

- ¥ Another issue related to authorization is the ability to access certain Admin pages in the web portal with a user role. This does not affect the pages where the above issues were found, but it indicates that future development of the system might introduce vulnerable pages that are accessible for any user. (24-4:11)

#### Configuration issues in the development process and server administration

- ¥ The source code in the git repository's commit history contains credentials. (24-4:12)
- ¥ When going through the setup process some outdated dependencies with known vulnerabilities were found. None of these were exploited during the test, but this indicates a problem in dependency management. (24-4:13)
- ¥ The application is then deployed on a server that has a single user which is shared by multiple developers. This makes it difficult to hold individual users accountable for their actions. (24-4:14)
- ¥ The user also has the permission to escalate their privileges to root by using sudo without a password. Best practice would be to require further authentication in order to escalate privileges. (24-4:15)

## 2. Introduction

A Data WareHouse (DWH) is a centralized system that accumulates and processes data from various sources across a company or organization. At Example, it is mainly used to combine data from projects and the human resources department. Thus, it is possible to obtain a quick overview about the current state of the employees' occupancy, project profitability, and general employee data.

The test took place as part of Example's efforts to constantly improve information security, both for internal and customer projects. Testing its internal systems is important for audit certification and general security improvement. The DWH represents such an internal system with high security requirements, as its centralized nature and sensitive data makes it a lucrative target for attacks.

The DWH was built by an internal team at Example. The current system replaced a previous version and has been under constant development since fall 2022. It is mainly written in python using the framework Dash for visualization of redacted pandas dataframes. The server runs on a Flatcar Linux VMware virtual machine inside the Example- VPN. For authentication, a Quay OAuth proxy is used together with a traefik proxy. OAuth is set up to use the Example Azure Entra ID rights and role management system. The data to visualize is fetched from an external database hosted on a Microsoft SQL server.

As a version control system, a repository in the Example- Gitlab is employed. Here, a pipeline was created for testing the code and dependencies, and building the provided docker images.

### 3. Organizational information

The pentesting team was formed of the following people:

*Table 1.1 Contact information*

Name	Role	Mobile	E-Mail
Jan Girlich	Pentesting Lead	+49 170 3748758	jan.girlich@iteratec.com
Johannes Merkert	Pentester	+49 170 3748528	johannes.merkert@iteratec.com
Sebastian Franz	Pentester	+49 170 3748686	sebastian.franz@iteratec.com

The timeline of the test was as follows:

*Table 1.2 Timeline*

Date	Event
26.03.2024	Initial meeting
14.05.2024	Kickoff
14.05.-21.05.2024	Pentesting
21.05-29.05.2024	Reporting
10.06.2024	Report finalization

## 4. Scope of the pentest

The pentest focused on two aspects of the DWH: The frontend web application and the underlying hosting server. The test was conducted as a white-box test with the source code available for the testers. The latest commit of the tested version was `82d7bae38ca3f7fa193f61697355e488ebf7fd43` [<https://git.example.com/DWHPY/-/commit/82d7bae38ca3f7fa193f61697355e488ebf7fd43>] from 14.05.24, 14:23.

The web interface was tested for unauthorized access, common web vulnerabilities such as XSS, CSRF or remote code execution, vulnerable up- or download management and vulnerable dependencies. ([OWASP Top Ten](https://owasp.org/www-project-top-ten/) [<https://owasp.org/www-project-top-ten/>]) On the host system, the linux and docker container configurations were investigated. The setup of the OAuth proxy that interacts with the DWH web application was also tested for common attacks or possible vulnerabilities. Finally, the server and repository were checked for leaked secrets and vulnerable secret management.

The following user roles were checked:

- ¥ User
- ¥ Developer
- ¥ HR
- ¥ GL
- ¥ REWE
- ¥ Admins

### 4.1. Limitations

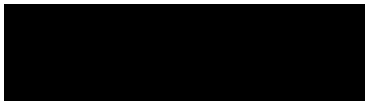
The pentest team did not face any problems during the test that hampered the test process.

## 5. Identified issues

All findings that were made during the pentest are listed in this section. Their criticality was scored using the Common Vulnerability Scoring System (CVSS), [version 3.1](https://www.first.org/cvss/calculator/3.1) [<https://www.first.org/cvss/calculator/3.1>].

### 5.1. Finding 24-4:1: Authenticated Remote Code Execution in ETL

Command injection in the ETL step definition is possible, leading to remote code execution (RCE). The code is executed with root privileges.



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

#### Prerequisites

The attacker must have the role Admins or Developer to be authorized to use the ETL function.

#### Description

During the security assessment, it was identified that there is a command injection vulnerability in the ETL ([ETL](https://dwh-dev.example.com/etl) [<https://dwh-dev.example.com/etl>]) steps definition.



*Figure 3.1 Button that triggers the etc process with the chosen steps.*

This vulnerability allows an attacker to inject arbitrary commands into the ETL process, leading to remote code execution.

The user-provided data is neither validated nor sanitized before being used. The vulnerability can be exploited by adding arbitrary commands to the values of the `run_steps_etl` property in the POST request for the ETL process.

*Listing 3.1 A POST request to the ETL process injecting the command i d*

```
Ê1 POST /_dash-update-component HTTP/2
Ê2 Host: dwh-dev.example.com
Ê3 Content-Length: 405
Ê4 Content-Type: application/json
Ê5 Origin: https://dwh-dev.example.com
Ê6 Referer: https://dwh-dev.example.com/etl
Ê7 Accept-Encoding: gzip, deflate, br
Ê8 {
Ê9   "output":
Ê  "etl_interval.disabled@9b5ec237f6a5b339a513d0e922629beba0de8d1482a87bdd1f8c
Ê  2c86fb5a67c1",
10   "outputs": {
11     "id": "etl_interval",
12     "property":
Ê  "disabled@9b5ec237f6a5b339a513d0e922629beba0de8d1482a87bdd1f8c2c86fb5a67c1"
13   },
14   "inputs": [
15     {
16       "id": "start_etl_button",
17       "property": "n_clicks",
18       "value": 1
19     }
20   ],
21   "changedPropIds": [
```

```
22     "start_etl_button.n_clicks"
23 ],
24 "state": [
25     {
26         "id": "run_steps_etl",
27         "property": "value",
28         "value": [
29             "1; id;"
30         ]
31     }
32 ]
33 }
```

The vulnerability exists because [backend for the ETL process](https://git.example.com/dwh/DWHPY/-/blob/a1d75f176d19986fe776c494c4211131dad7496d/dwh/etl/etl_subprocess.py) [https://git.example.com/dwh/DWHPY/-/blob/a1d75f176d19986fe776c494c4211131dad7496d/dwh/etl/etl\_subprocess.py] calls the bash script [command.sh](https://git.example.com/dwh/DWHPY/-/blob/a1d75f176d19986fe776c494c4211131dad7496d/scripts/command.sh) [https://git.example.com/dwh/DWHPY/-/blob/a1d75f176d19986fe776c494c4211131dad7496d/scripts/command.sh] and uses user-controlled data as arguments.

*Listing 3.2 Call of the ETL to the bash script command.sh in the source code*

```
69     def run(self):
70         try:
71             basecmd = ["../scripts/command.sh"]
72             cmd_with_args = list(iteration_utilities.flatten([basecmd,
73 sorted(self.__arguments)]))
74             self.__subprocess = subprocess.Popen(
75                 cmd_with_args, shell=False, stdout=subprocess.PIPE,
76                 stderr=subprocess.STDOUT
77             )
```

Because the gunicorn appserver is running as root user ([see finding 24-4:3](#)), the injected code is executed with root privileges. This significantly increases the risk and potential impact of the attack.

*Figure 3.2 Output of the RCE in the ETL logs showing that the injected id was executed and ran with root privileges.*

## Impact

- ¥ Remote Code Execution (RCE): An attacker can inject arbitrary commands that will be executed during the ETL process. This can lead to the execution of malicious code on the server.
- ¥ Privilege Escalation: Since the code is executed with root privileges, an attacker can gain full control over the system. This includes modifying system files, installing malware, and accessing sensitive data.
- ¥ Data Breach: An attacker may access and exfiltrate sensitive data processed during the ETL steps, leading to potential data breaches and loss of confidentiality.
- ¥ Service Disruption: Malicious commands can disrupt the ETL process, leading to data processing failures and service outages, which can impact business operations.
- ¥ Full System Compromise: With root access, an attacker can pivot to other parts of the network, install persistent backdoors, and compromise additional systems, leading to widespread security incidents.

## Recommendations

- ¥ Input Validation and Sanitization: Instead of passing user-controlled data as arguments to scripts or programs, consider implementing defined methods or APIs that handle different cases and scenarios. Ensure that all inputs to the ETL process are properly validated and sanitized to prevent command injection. Use safe functions and libraries that automatically handle escaping of special characters.
- ¥ Use Least Privilege Principle: Run the ETL processes with the minimum necessary

privileges. Avoid running ETL steps with root privileges. Create a dedicated user with restricted permissions for executing ETL steps.


- ¥ Environment Hardening: Harden the environment where the ETL processes run by implementing security controls such as SELinux, AppArmor, and seccomp profiles to limit the impact of potential exploits.

## Further Information

- ¥ <https://snyk.io/de/blog/command-injection-python-prevention-examples/>

## 5.2. Finding 24-4:2: ETL Page Contains Execution Logs

ETL step execution shows logs in the webapp containing information about the system.



Attack Vector	Network
Attack Complexity	Low
Privileges Required	High
User Interaction	None
Scope	Unchanged
Confidentiality	High
Integrity	None
Availability	None

### Prerequisites

The attacker must be authorized to use the ETL function.

### Description

During the security assessment, it was identified that the **ETL page** [<https://dwh-dev.example.com/etl>] displays the logs of ETL executions. These logs contain detailed information about the system. Although the page is only accessible to admin users, displaying sensitive system information in logs can still pose a security risk.



*Figure 4.1 ETL logs displaying database server information*

## Impact

- ¥ Information Disclosure: The logs may contain sensitive information about the system, such as file paths, database queries, configuration details, and error messages. This information can be valuable to attackers if they gain access to it, even indirectly.
- ¥ Admin Account Compromise: If an admin account is compromised, the attacker can view the ETL execution logs and gather detailed system information that can be used to further exploit the system or other connected systems.
- ¥ Internal Reconnaissance: Detailed logs can provide a roadmap for attackers, aiding in internal reconnaissance and allowing them to identify potential weaknesses and vulnerabilities in the system.

## Recommendations

- ¥ Do not display logs in the web application: Perform debugging and log analysis directly on the server. Implement robust logging mechanisms and use secure protocols to access log files for debugging purposes. This practice helps maintain the confidentiality and integrity of system information and reduces the attack surface.
- ¥ Log Sanitization: Ensure that logs displayed on the ETL page are sanitized to remove any sensitive information. Only include necessary details that are useful for debugging and monitoring purposes without exposing system internals.
- ¥ Limit Log Detail: Avoid logging excessively detailed system information. Ensure that logs contain only the essential information required for operational purposes. Use defined error and success messages instead of log messages.

### 5.3. Finding 24-4:3: Gunicorn Application Server Runs as Root Process

The Gunicorn app server runs with root privileges, which is more than necessary and allows attacks on other system components.



Attack Vector	Local
Attack Complexity	Low
Privileges Required	High
User Interaction	None
Scope	Unchanged
Confidentiality	High
Integrity	High
Availability	High

#### Prerequisites

The attacker must be able to perform RCE on the webapp.

#### Description

During the security assessment, it was observed that the Gunicorn application server is running as the root user. Running services as the root user poses significant security risks. It increases the attack surface when the server is compromised, potentially leading to a full system compromise. If exploited, this could lead to unauthorized access, data breaches, and potential control over the entire server by malicious actors.

Figure 5.1 Gunicorn running as root



## Impact

- ¥ Privilege Escalation: If an attacker exploits a vulnerability within the Unicorn application, they could gain root access to the system, leading to full control over the virtual machine or container. This allows the attacker to perform any actions, including installing malware, altering system configurations, and accessing sensitive data.
- ¥ Host System Compromise: Running as root increases the risk of container escape, where an attacker could potentially gain access to the host system from within the container.
- ¥ Increased Attack Surface: Root privileges grant extensive permissions, increasing the potential damage an attacker can inflict. This includes the ability to modify or delete system files, change network configurations, and access other sensitive services running on the host.

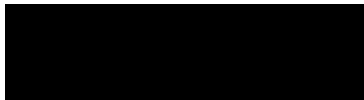
## Recommendations

- ¥ Run as a Non-Root User: Configure the Unicorn application server to run as a non-root user.
- ¥ Least Privilege Principle: Ensure the non-root user has the minimum necessary permissions to run the application. Avoid granting unnecessary capabilities. This significantly reduces the risk of privilege escalation and limits the potential impact of a security breach.



## 5.4. Finding 24-4:4: Docker Container User is Root

The DWH docker container user is root, which allows executing arbitrary code and escaping the container.



Attack Vector	Local
Attack Complexity	Low
Privileges Required	High
User Interaction	None
Scope	Unchanged
Confidentiality	High
Integrity	High
Availability	High

### Prerequisites

The attacker can access the DWH container.

### Description

During the security assessment of the DWH Docker container, it was observed that the default user is set to root. This practice poses a significant security risk and should be addressed immediately. Running containers as the root user grants extensive privileges that, if exploited, could lead to severe security breaches.

By entering a shell in the running container, it can be quickly verified if the user is set to root:

```
core@dwH-qa ~ $ docker exec -it 5099076b74c4 /bin/bash
root@dwH: /app/dwH#
```

### Impact

¥ Privilege Escalation: If an attacker exploits a vulnerability within the application running in the container, they could gain root access to the container. From there, they

might attempt to escape the container environment to gain control of the host system.

- ¥ Host System Compromise: Containers running as root have more access to the host system, especially when privileged or with certain capabilities enabled. This increases the risk of the container compromising the host.
- ¥ Violation of Least Privilege Principle: Security best practices advocate for the principle of least privilege, where applications and services should run with the minimum level of access required. Running containers as root violates this principle.

## Recommendations

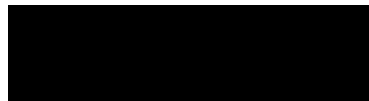
- ¥ Run as Non-Root User: Modify the Dockerfile to specify a non-root user.
- ¥ Least Privilege Principle: Ensure that the non-root user has the minimum necessary permissions to run the application effectively. Avoid granting additional capabilities unless absolutely necessary.

## Further Information

- ¥ CIS Docker Benchmark v.1.6.0 [<https://www.cisecurity.org/benchmark/docker>]

## 5.5. Finding 24-4:5: Docker Socket is Mounted in Container

The Docker socket is mounted in multiple containers which may enable an attacker to escape the container.



Attack Vector	Local
Attack Complexity	Low
Privileges Required	High
User Interaction	None
Scope	Changed
Confidentiality	High
Integrity	High
Availability	High

### Prerequisites

The attacker can execute arbitrary code in container and is able to run docker container.

### Description

During the security assessment, it was discovered that the Docker socket (/var/run/docker.sock) is mounted inside the container's traefik and dwhapp. The Docker socket provides access to the Docker API, which can be used to manage Docker containers and services on the host machine.

Figure 7.1 Docker socket is mounted and readable

The mount is configured in the `docker-compose.yml` [<https://git.example.com/dwh/DWHPY/-/blob/a1d75f176d19986fe776c494c4211131dad7496d/docker-compose.yml>].

Listing 7.1 Socket mount in traefik configuration

```
3 image: traefik:v3.0
```

```
4   ports:
5     - "80: 80"
6     - "443: 443"
7   volumes:
8     - "/var/run/docker.sock:/var/run/docker.sock: ro"
9     - ". /docker/traefik/config:/etc/traefik"
10    - ". /docker/traefik/work:/var/run/traefik"
```

*Listing 7.2 Socket mount in dwhapp configuration*

```
73   dwhapp:
74     build:
75       context: .
76       dockerfile: docker/dwhapp/Dockerfile
77     image: "${DOCKER_REGISTRY_REF}dwh/dwhpy/app:${DOCKER_IMAGE_VERSION}"
78     volumes:
79       - ". /docker-compose.yml:/app/docker-compose.yml"
80       - ". /.env:/app/.env"
81       - "/var/run/docker.sock:/var/run/docker.sock"
```

The mount in the traefik container is read-only, which prevents write access to the host. The mount in the dwhapp container, on the other hand, is not limited to read-only, which allows full write access to the host system.

To leverage the vulnerability, run a docker container inside the docker container that has the docker socket mounted. There are three main methods:

1. Mount the host disk as container file system root to gain full read and write access on the host's file system:

```
docker run -it -v /:/host/ ubuntu:18.04 chroot /host/ bash
```

2. Run a container with `--privileged` flag and use the host's namespace. Then enter the namespaces of the process with pid 1 (init process) to spawn a shell to gain full access to the host:

```
root@dwh:/app/dwh# docker run -it --rm --pid=host --privileged ubuntu bash
root@875131f22cb3:/# nsenter --target 1 --mount --uts --ipc --net --pid --
bash
```

- ! --mount: Enter the mount namespace.
- ! --uts: Enter the UTS (Unix Time-sharing System) namespace.
- ! --ipc: Enter the IPC (Inter-Process Communication) namespace.
- ! --net: Enter the network namespace.
- ! --pid: Enter the PID namespace.

3. Run a container without --privileged flag, mount the host disk as container file system root and use the host's namespaces to gain full access to the host:

```
root@dwh:/app/dwh# docker run -it -v /:/host/ --cap-add=ALL --security-opt  
apparmor=unconfined --security-opt seccomp=unconfined --security-opt  
label:disable --pid=host --userns=host --uts=host --cgroupns=host ubuntu  
chroot /host/ bash
```

- ! --cap-add=ALL: Adds all kernel capabilities to the container, effectively giving it root-like privileges.
- ! --security-opt apparmor=unconfined: Disables AppArmor, a Linux kernel security module, for the container. This allows the container to bypass AppArmor's restrictions.
- ! --security-opt seccomp=unconfined: Disables seccomp, a Linux kernel security feature, for the container. This allows the container to execute system calls without restriction.
- ! --security-opt label:disable: Disables SELinux (Security-Enhanced Linux) labeling for the container. This allows the container to access resources without SELinux restrictions.
- ! --pid=host: Tells Docker to use the host's PID namespace, which allows the container to see and interact with the host's processes.
- ! --userns=host: Tells Docker to use the host's user namespace, which allows the container to access and manipulate the host's user accounts and permissions.
- ! --uts=host: Tells Docker to use the host's UTS (Unix Time-sharing System) namespace, which allows the container to access and manipulate the host's hostname and domainname.
- ! --cgroupns=host: Tells Docker to use the host's cgroup namespace, which allows the container to access and manipulate the host's cgroup settings.

This configuration poses a significant security risk as it can potentially allow an attacker to escape the container and gain control over the host system.

*Figure 7.2 From a root shell in the Docker container, the attack leads to a root shell on the host system.*

## Impact

- ¥ Container Escape: An attacker with access to the Docker socket inside the container can use the Docker API to start new containers, which can run with elevated privileges or access sensitive parts of the host filesystem. This can lead to a full container escape and compromise of the host system.
- ¥ Privilege Escalation: The Docker API can be used to run commands as the root user on the host, effectively allowing an attacker to escalate privileges and perform any actions that the root user can.
- ¥ Unauthorized Access: If an attacker gains control of the Docker socket, they can access and manipulate other containers running on the same host. This includes reading sensitive data, stopping critical services, and injecting malicious code.
- ¥ Service Disruption: Manipulating the Docker environment can lead to Denial of Service attacks, where the attacker stops or disrupts running containers and services, leading to potential outages and service disruptions.

## Recommendations

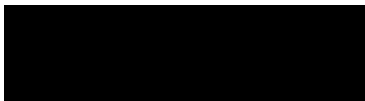
- ¥ Avoid Mounting the Docker Socket: Do not mount the Docker socket inside containers unless absolutely necessary. Evaluate alternative approaches to achieve the required functionality without exposing the Docker socket.
- ¥ Use Docker API with Proper Authentication: If access to the Docker API is required, use proper authentication and authorization mechanisms. Ensure that the API is exposed over a secure channel and restrict access to trusted entities.
- ¥ Implement Least Privilege Principle: Ensure that containers run with the minimum required privileges. Avoid running containers with `--privileged` or excessive capabilities. Use specific capabilities only when necessary.
- ¥ Use Namespaces and Cgroups: Leverage Docker's namespace and cgroup isolation features to limit the scope and impact of a potential container breakout. These features provide an additional layer of security by isolating containers from the host and from each other.

## Further Information

- ¥ [HackTricks: Docker Breakout / Privilege Escalation](https://book.hacktricks.xyz/linux-hardening/privilege-escalation/docker-security/docker-breakout-privilege-escalation) [https://book.hacktricks.xyz/linux-hardening/privilege-escalation/docker-security/docker-breakout-privilege-escalation]
- ¥ [CIS Docker Benchmark v.1.6.0](https://www.cisecurity.org/benchmark/docker) [https://www.cisecurity.org/benchmark/docker]

## 5.6. Finding 24-4:6: Secrets in Environment File

The host systems' environment files at `/home/core/dwh/.env` contain plain text secrets.



Attack Vector	Local
Attack Complexity	Low
Privileges Required	High
User Interaction	None
Scope	Changed
Confidentiality	Low
Integrity	Low
Availability	Low

### Prerequisites

The attacker must be inside Example's internal network and authorized to be able to access the dwh machine.

### Description

The environment files located on the host machines `dwh-dev.example.com`, `dwh-qa.example.com`, and `dwh.example.com` at `/home/core/dwh/.env` contain plain text secrets, including database usernames and passwords, client secrets, and mail user secrets.



*Figure 8.1 Content of the .env file including secrets*

The file permissions are set to be readable by all users.

*Figure 8.2 World-readable .env file*

If an attacker gains access to this information, they can use it to access and attack connected systems and resources.

## Impact

¥ Unauthorized access: If an attacker gains access to the environment file, they can

easily retrieve sensitive secrets. This could lead to unauthorized access to databases, APIs, and other critical services.

- ¥ Data breach: Compromised secrets can result in data breaches, where sensitive information is accessed, modified, or exfiltrated. This can have severe legal and financial repercussions.
- ¥ Lateral movement: Attackers with access to plaintext secrets can leverage them to move laterally within the network, compromising additional systems and services.
- ¥ Email phishing attacks: The compromised SMTP secrets can be used to create phishing emails that appear to be from a trustworthy internal sender, potentially leading to further security incidents.

## Recommendations

- ¥ Use a secret management tool: Implement a secrets management solution such as HashiCorp Vault to securely store and manage sensitive secrets.
- ¥ Access controls: Restrict access to the `.env` file using appropriate file permissions. Ensure only the necessary application processes and users have read access to this file.
- ¥ Rotate credentials: Rotate database credentials, client secrets, and mail user secrets to minimize the impact of a potential breach.

## Further Information

- ¥ **Vault** [<https://www.vaultproject.io/>]

## 5.7. Finding 24-4:7: Linked Databases with Financial Data in MSSQL

The DWH MSSQL database is linked to another SQL database server. This server contains sensitive financial data about Example.



Attack Vector	Local
Attack Complexity	Low
Privileges Required	High
User Interaction	None
Scope	Changed
Confidentiality	High
Integrity	High
Availability	None

### Prerequisites

The attacker must have access to the MSSQL database server.

### Description

The DWH MSSQL database is linked to another SQL database server. Linked databases enable queries that combine data from multiple external sources. This feature could be leveraged to execute queries on remote database servers without explicit authentication.



*Figure 9.1 Linked sql database server*

In this scenario, the user did not have privileges to execute commands on the machine but could access the databases on the server and view the data within relying solely on the MSSQL credentials found in the .env file (see [finding 24-4:6](#)). These databases contain potentially sensitive financial data related to Example, covering the period from 2020 up to June 2024.

*Figure 9.2 The linked database stores financial data from February 2020 to June 2024.*

This linkage exposes potentially sensitive financial information to unauthorized access.

## Impact

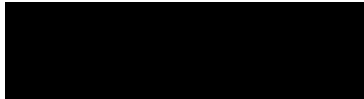
- ¥ Data Exposure: The linkage of databases increases the risk of unauthorized access to sensitive financial data. If the DWH MSSQL database is compromised, the attacker could potentially access the linked database and the financial information it contains.
- ¥ Security Boundary Weakening: Linking databases can weaken the security boundaries between different data sets. Sensitive financial data should be isolated and protected with strict access controls.
- ¥ Compliance Risks: Handling financial data requires adherence to various regulatory and compliance standards. Inadequate protection of this data could lead to non-compliance and potential legal repercussions.

## Recommendations

- ¥ Review and Restrict Access: Conduct a thorough review of the necessity for linking the databases. Restrict access to only those users and systems that absolutely require it. Ensure that proper authentication and authorization mechanisms are in place.
- ¥ Implement Strong Access Controls: Apply robust access controls to the linked databases. Use roles and permissions to ensure that only authorized users can access the sensitive financial data.
- ¥ Monitor and Audit: Implement logging and monitoring to track access to the linked databases. Regularly audit these logs to detect and respond to any unauthorized access attempts.
- ¥ Data Segregation: Where possible, segregate the sensitive financial data from other datasets. Consider using dedicated database servers for sensitive information to enhance security.
- ¥ Encryption: Ensure that data at rest and in transit is encrypted. Use industry-standard encryption methods to protect sensitive financial data from unauthorized access.

## 5.8. Finding 24-4:8: Authenticated Remote Code Execution in MSSQL

Command injection in MSSQL database is possible due to excessive permissions of the user SA. This leads to Remote Code Execution (RCE).



Attack Vector	Local
Attack Complexity	Low
Privileges Required	Low
User Interaction	None
Scope	Changed
Confidentiality	Low
Integrity	Low
Availability	Low

### Prerequisites

The attacker must have access to the MSSQL database server with a user account that has permissions to activate the xp\_cmdshell procedure.

### Description

A command injection vulnerability was identified in the DWH MSSQL Server 2019 configurations of dwh-dev-master.example.com, dwh-qa-master.example.com, and dwh.example.com.

An attacker with access to the database can activate the xp\_cmdshell procedure and execute arbitrary commands, leading to Remote Code Execution. The vulnerability exists because the MSSQL database configuration allows xp\_cmdshell procedure to be activated and the user with the credentials found in the .env file (see finding 24-4:6) has the necessary permissions to do this.

*Figure 10.1 Activating the xp\_cmdshell procedure*

Running `whoami` gives details about the user context and privileges with which the commands are executed on the server.

*Figure 10.2 Executing whoami via xp\_cmdshell*

The output of `whoami /priv` shows that the commands are executed as the user `nt service\mssqlserver`. This account is a service account with only limited privileges. The privileges interesting from an attacker point of view are the following:

- ¥ `SeManageVolumePrivilege`: Allows the service to perform volume maintenance tasks, which can be used to manipulate disk volumes.



¥ **SeImpersonatePrivilege**: Allows the service to impersonate clients after authentication, which can be used to access resources and data without proper authorization.

An attacker possibly can use this privileges to gain access to data and resources without further authorization. There are ready-to-use exploits available to leverage these vulnerabilities (see Further Information).

## Impact

¥ **Remote Code Execution (RCE)**: An attacker can inject arbitrary commands that will be executed on the system. This can lead to the execution of malicious code on the server.

¥ **Privilege Escalation**: An attacker can use the enabled privileges to escalate their privileges further, potentially gaining administrative access to the system.

¥ **Data Breach**: An attacker can use the **SeImpersonatePrivilege** or **SeManageVolumePrivilege** privilege to access sensitive data and resources without proper authorization, leading to a potential data breach.

¥ **System Instability**: An attacker can use the **SeManageVolumePrivilege** privilege to manipulate disk volumes, potentially leading to system instability and data loss.

¥ **Full System Compromise**: An attacker can use an exploit for the **SeImpersonatePrivilege**, potentially leading to a full system compromise.

## Recommendations

¥ **Secure Configuration**: Restrict access to **xp\_cmdshell** (and other procedures) to only necessary users. Implement secure configuration and hardening of the MSSQL database, including restricting privileges and access to sensitive system resources (see [CIS MSSQL Server 2019 Benchmark](https://www.cisecurity.org/benchmark/microsoft_sql_server) [https://www.cisecurity.org/benchmark/microsoft\_sql\_server] for hardening recommendations).

¥ **Privilege Reduction**: Reduce the privileges of the MSSQL Server service account to the minimum necessary, revoking unnecessary privileges such as **SeImpersonatePrivilege**, **SeManageVolumePrivilege**, and **SeCreateGlobalPrivilege**. Implement a least privilege model to restrict access to sensitive resources and data.

¥ **Access Control and Segregation**: Implement access control mechanisms to segregate sensitive data and resources from the MSSQL Server service account. Restrict access to sensitive areas of the system, such as system files, directories, and registry settings.

## Further information

¥ **xp\_cmdshell procedure** [https://learn.microsoft.com/en-us/sql/relational-databases/system-stored-]

procedures/xp-cmdshell-transact-sql?view=sql-server-ver16]

- ¥ **Windows privileges** [<https://learn.microsoft.com/en-us/windows/win32/secauthz/privilege-constants>]
- ¥ **CIS Microsoft SQL Server 2019 Benchmark v 1.4.0** [[https://www.cisecurity.org/benchmark/microsoft\\_sql\\_server](https://www.cisecurity.org/benchmark/microsoft_sql_server)]
- ¥ **RottenPotato, Windows exploit for SeImpersonatePrivilege** [<https://github.com/breenmachine/RottenPotatoNG>]
- ¥ **Juicy-Potato, Windows exploit for SeImpersonatePrivilege** [<https://github.com/ohpe/juicy-potato>]
- ¥ **Rogue-Potato, Windows exploit for SeImpersonatePrivilege** [<https://k4sth4.github.io/Rogue-Potato/>]

## 5.9. Finding 24-4:9: Insecure Authorization from JWT

The way how a user and its role are extracted from the request's JWT is not secure.



Attack Vector	Local
Attack Complexity	Low
Privileges Required	High
User Interaction	None
Scope	Unchanged
Confidentiality	High
Integrity	Low
Availability	None

### Prerequisites

To abuse the vulnerability, one needs access to the host server.

### Description

The DWH application uses OAuth2 via Microsoft for authentication. Every un-authenticated request is redirected to a *quay oauth proxy* server via the installed *traefik proxy*. The *oauth proxy* handles the login procedure and creates both an *oauth2\_proxy Cookie* and a *JWT*. The *traefik proxy* then updates the request with the *JWT* and sends it to the DWH application. It includes authorization information from the Example AD/Entra ID. This information is extracted in the DWH and used to determine, which pages and data a user may view.

Figure 11.1 Sequence diagram of the authorization flow

The following source snippet contains the `extract_user_from_request` function in the `dwh/frontend/auth/auth.py` script [https://git.example.com/dwh/DWHPY/-/blob/82d7bae38ca3f7fa193f61697355e488ebf7fd43/dwh/frontend/auth/auth.py].

It contains several issues which are marked with a # ! comment:

Listing 11.1 `auth.py`

```
24 def extract_user_from_request():
25     bearer_token = request.headers.get("Authorization")
26     if bearer_token:
27         # ! Could be improved by checking if "Bearer" is in bearer_token
28         # and split accordingly
29         jwt_token = bearer_token[7:]
30         # ! Signature is not verified: This enables an attacker to
31         # modify the jwt in any way, including the given roles
32         decoded_jwt_token = jwt.decode(jwt_token,
33         options={"verify_signature": False})
34         # ! Sensitive information like a (decoded) JWT should not be
35         # stored in the logs
36         __logger.debug("JWT Token: %s", decoded_jwt_token)
37         name = decoded_jwt_token.get("name", "")
38         email = decoded_jwt_token.get("email", "")
39         # merge roles through group assignment (dwh_roles) and roles with
40         # direct role assignment (roles)
41         roles = get_roles_from_jwt(decoded_jwt_token)
42         id = get_mitarbeiter_id_from_email(email)
43         # ! If the Authorization header was somehow removed from the
```

```

39     request, the user gets admin rights in worst case
40     else:
41         name = "Developer"
42         email = "me@example.com"
43         id = 1628 # my intern_bk
44         # default roles for testing, change at your will
45         # empty list for avoiding exceptions
46         if config.env == "production":
47             roles = []
48         elif config.env.endswith("admin"):
49             roles = ["Admins"]
50         else:
51             roles = ["Public"]
52     return User(name, email, roles, id)
```

From the snippet follows, that removing the token from the request to the DWH application or changing the header key creates a fallback user which has admin rights in the worst case. Additionally, it is possible to change the JWT to contain the *Admins* role, because the JWT signature is not verified.

Both issues can be exploited by manipulating the *traefik configuration* file at [docker/traefik/config/dynamic.yml](https://git.example.com/dwh/DWHPY/-/blob/82d7bae38ca3f7fa193f61697355e488ebf7fd43/docker/traefik/config/dynamic.yml) [https://git.example.com/dwh/DWHPY/-/blob/82d7bae38ca3f7fa193f61697355e488ebf7fd43/docker/traefik/config/dynamic.yml].

¥ Removing the JWT by changing the header key:

Listing 11.2 *dynamic.yml*

```

61     oauth-auth-redirect:
62     forwardAuth:
63         address: http://oauth:{{env "OAUTH_PROXY_PORT"}}
64         trustForwardHeader: true
65         authResponseHeaders:
66             - Authorization123 # Set to any name other than Authorization
```

This makes the user become Developer:

Figure 11.2 User Developer in the web interface, as shown by the greeting in the top right corner.

¥ Manipulating the JWT:

a) Get the JWT from the **oauth proxy** [https://oauth.dwh-dev.example.com/] via a GET request.

Listing 11.3 GET request to fetch a JWT from the OAuth proxy

```

1 GET / HTTP/1.1
2 Host: oauth.dwh-dev.example.com
3 Cookie: _oauth2_proxy=djlu..
4 Sec-Ch-Ua: "Not-A.Brand";v="99", "Chromium";v="124"
5 Sec-Ch-Ua-Mobile: ?0
6 Sec-Ch-Ua-Platform: "Linux"
7 Upgrade-Insecure-Requests: 1
8 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36
  (KHTML, like Gecko) Chrome/124.0.6367.118 Safari/537.36
9 Accept:
  text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp
  ,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7
10 Sec-Fetch-Site: none
11 Sec-Fetch-Mode: navigate
12 Sec-Fetch-User: ?1
13 Sec-Fetch-Dest: document
14 Accept-Encoding: gzip, deflate, br
15 Accept-Language: de-DE,de;q=0.9,en-US;q=0.8,en;q=0.7
16 Priority: u=0, i
17 Connection: close

```

Listing 11.4 Response with the JWT

```

1 HTTP/2 202 Accepted
2 Authorization: Bearer eyJ0..
3 Content-Type: text/plain; charset=utf-8
4 Date: Thu, 23 May 2024 09:07:48 GMT
5 Gap-Auth: sebastian.franz+dwh-hr@iteratec.com
6 Strict-Transport-Security: max-age=315360000; includeSubDomains; preload
7 X-Content-Type-Options: nosniff
8 X-Frame-Options: DENY
9 X-Xss-Protection: 1; mode=block

```

```
10 Content-Length: 13
11
12 Authenticated
```

b) Decode the non-signature part of the JWT and change the current role to "dwh\_roles": "Admins".

c) Change the *traefik* configuration to include the manipulated JWT instead:

Listing 11.5 *dynamic.yml*

```
19     dwhapp-route:
20         rule: "Host(`{{env \"DWH_URL\"}}`)"
21         tls:
22             certResolver: "{{env \"CERT_RESOLVER\"}}"
23         middlewares:
24             - oauth-auth-redirect
25             - custom-auth-header # Add the new middleware to dwhapp-route
26         service: dwhapp-service
27         entryPoints:
28             - websecure
```

Listing 11.6 *dynamic.yml*

```
61     oauth-auth-redirect:
62         forwardAuth:
63             address: http://oauth:{{env \"OAUTH_PROXY_PORT\"}}
64             trustForwardHeader: true
65             authResponseHeaders:
66                 - Authorization123 # Set to any name other than Authorization
67         custom-auth-header:
68             headers:
69                 customRequestHeaders:
70                     Authorization: "Bearer eyJ0eXAiOiI...\" # Manipulated Token with
71                     Admins role
```

For both methods to work, the *traefik* container needs to be restarted after changing the configuration:

```
$ docker compose up -d --no-deps --force-recreate --build proxy
```

## Impact

- ¥ Data Breach: Gaining unauthorized admin rights in the application allows an attacker to view very sensitive information. This might lead to financial damage or reputation loss if business figures are published.
- ¥ Unauthorized Access: The data also includes information about every employee at Example. It could be abused for doxing, phishing, or enticement.

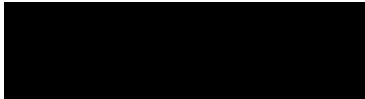
## Recommendations

- ¥ Code Fixing: The commented sections in the authorization function above should be fixed. Especially verifying the signature of the JWT is crucial to avoid token manipulation.
- ¥ User Rights Management: Furthermore, changing configuration files or containers on the server should require admin rights (see finding 24-4:15).
- ¥ Hiding JWT from Requests to OAuth Proxy: It should also be checked if it is possible to hide the JWT from responses to requests that go to <https://oauth.dwh-dev.example.com/> and are not coming from the DWH app directly.



## 5.10. Finding 24-4:10: Incorrect Handling of Preflight Request Leads to OAuth Redirect

The server incorrectly handles an OPTIONS preflight request by issuing a 302 redirect response to an OAuth 2.0 authorization URL.



Attack Vector	Network
Attack Complexity	Low
Privileges Required	None
User Interaction	None
Scope	Unchanged
Confidentiality	None
Integrity	None
Availability	None

### Prerequisites

The attacker must be inside Example's internal network.

### Description

Following is an OPTIONS preflight request. The expected response would be the appropriate CORS headers.

```
OPTIONS /_dash-update-component HTTP/2
Host: example.com
Accept: */*
Access-Control-Request-Method: POST
Access-Control-Request-Headers: content-type
Origin: null
Sec-Fetch-Mode: cors
Sec-Fetch-Site: cross-site
Sec-Fetch-Dest: empty
Accept-Encoding: gzip, deflate, br
Accept-Language: en-GB, en-US; q=0.9, en; q=0.8
```

Priority: u=1, i

Instead the response is a 302 redirect:

```
HTTP/2 302 Found
Cache-Control: no-cache, no-store, must-revalidate, max-age=0
Content-Type: text/html; charset=utf-8
Date: Fri, 31 May 2024 09:59:15 GMT
Expires: Thu, 01 Jan 1970 01:00:00 CET
Location: https://login.microsoftonline.com/...
Set-Cookie: _oauth2_proxy_csrf=...; Path=/; Domain=example.com; Expires=Fri,
31 May 2024 10:14:15 GMT; HttpOnly
X-Accel-Expires: 0
Content-Length: 446

<a href="https://login.microsoftonline.com/oauth2/v2.0/authorize?... ">Found</a>.
```

This is not the expected behavior for a preflight request and indicates a misconfiguration that could lead to unintended behavior or security issues.

## Impact

- ¥ Authentication Disruption: Redirecting preflight requests to an OAuth login page disrupts the intended flow of authentication. This can result in failed requests and a poor user experience, making the application less reliable and harder to use.
- ¥ Security Policy Bypass: Misconfigured preflight handling can allow attackers to bypass security policies intended to restrict cross-origin requests. This can lead to unauthorized access to resources, potentially exposing sensitive data.
- ¥ Unintended Exposure of Sensitive Data: The redirection exposes details about the authentication process, such as the OAuth authorization URL and state parameters. Attackers can use this information to understand the authentication flow for further exploitation.

## Recommendations

Ensure that the server correctly handles OPTIONS preflight requests by returning the appropriate CORS headers without redirection.

## Further Information

¥ [CORS Explanation](https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS) [https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS]

## 5.11. Finding 24-4:11: Pages in Admin Section Accessible for Non-Admin Users

Pages in the admin section are accessible for normal (non-admin) users.



Attack Vector	Network
Attack Complexity	Low
Privileges Required	Low
User Interaction	None
Scope	Unchanged
Confidentiality	Low
Integrity	None
Availability	None

### Prerequisites

The attacker must be authorized to access the DWH application.

### Description

Pages located in the admin section of the application are accessible to normal (non-admin) users. This concerns the following pages:

```
https://example.com/upload
https://example.com/download
https://example.com/etl
https://example.com/data_validation
https://example.com/weblog
```

The functionality of these pages is mostly broken, except for two pages:

¥ For the **download** [https://example.com/download] page, content from the *Channel* dropdown menu can be selected:



*Figure 13.1 Download channels selection*

Uploading and downloading was not possible for non-admin users during our tests.

- ¥ The **weblog** [<https://example.com/weblog>] page shows the number of accesses per page, even for non-admin users. While the content of this page does not appear to be critical, its placement in the admin section indicates that it should likely be restricted to admin users only.

## Impact

- ¥ Unauthorized Access: Allowing non-admin users to access pages within the admin section can lead to unauthorized access to information and functionalities that should be restricted.
- ¥ Access Control Bypass: The presence of such a misconfiguration suggests that there may be other access control issues within the application, potentially allowing further unauthorized access to sensitive areas.
- ¥ Information Leakage: Although the current content of the "Zugriffsstatistik" page is not critical, future changes or updates to this page could introduce sensitive information that non-admin users should not see.

## Recommendations

- ¥ Restrict Access to Admin Pages: Ensure that all pages and functionalities within the admin section are properly restricted to admin users only. Implement role-based access control (RBAC) to enforce this restriction.

## 5.12. Befund 24-4:12: Credentials in Git repository

Das Git-Repository **DWHPY** [<https://git.example.com/dwh/DWHPY>] enthält (veraltete) Geheimnisse.



Attack Vector	Network
Attack Complexity	Low
Privileges Required	Low
User Interaction	None
Scope	Unchanged
Confidentiality	None
Integrity	None
Availability	None

### Voraussetzungen

Der/die Angreifer:in muss Zugriff auf das Gitlab-Repository haben.

### Beschreibung

Während der Sicherheitsüberprüfung wurde entdeckt, dass das **DWHPY** [<https://git.example.com/dwh/DWHPY>] Gitlab-Repository Geheimnisse unter Verwendung des Tools **Gitleaks** [<https://github.com/gitleaks/gitleaks>] enthält. Gitleaks ist ein Open-Source-Tool, das entwickelt wurde, um fest codierte Geheimnisse wie API-Schlüssel, Passwörter, private Schlüssel und andere sensible Informationen innerhalb der Versionskontroll-Historie eines Repositories zu erkennen. Obwohl die Geheimnisse veraltet zu sein scheinen und im aktuellen Zustand des Repositories (HEAD) nicht sichtbar sind, können sie dennoch ohne großen Aufwand in der Repository-Historie gefunden werden.

```
$ gitleaks detect --source=. --report-path=./dwh_gitleaks_report.json --report-format=json
```

Mögliche relevante Funde aus dem Gitleaks-Bericht:

1 [
-----

```
Ê2 {
Ê3   "Description": "Detected a Generic API Key, potentially exposing access
Ê   to various services and sensitive operations.",
Ê4   [...]
Ê5   "Match": "api_key=' b81e9xxxxxxxxxxxxxxxxxxxxfd7a87' ",
Ê6   "Secret": "b81e9xxxxxxxxxxxxxxxxxxxxfd7a87",
Ê7   "File": "dwh/ai/vanna_app.py",
Ê8   "SymLinkFile": "",
Ê9   "Commit": "841a0965d61328239c3883b601dca3980b0e2c22",
10   [...]
11 },
12 {
13   "Description": "Detected a Generic API Key, potentially exposing access
Ê   to various services and sensitive operations.",
14   [...]
15   "Match": "CLIENT_ID=1e875xxxxxxxxxxxxxxxxxxxxxxf5bf4",
16   "Secret": "1e875xxxxxxxxxxxxxxxxxxxxxxf5bf4",
17   "File": ".env",
18   "SymLinkFile": "",
19   "Commit": "527f44bec921d84dae953694cebfc2b33b1f14ba",
20   [...]
21 },
22 {
23   "Description": "Detected a Generic API Key, potentially exposing access
Ê   to various services and sensitive operations.",
24   [...]
25   "Match": "SECRET=fG-l zxxxxxxxxxxxxxxxxxxxxxxxxxxxxtf-54=",
26   "Secret": "fG-l zxxxxxxxxxxxxxxxxxxxxxxxxxxxxtf-54=",
27   "File": ".env",
28   "SymLinkFile": "",
29   "Commit": "527f44bec921d84dae953694cebfc2b33b1f14ba",
30   [...]
31 }
32 ]
```

## Auswirkungen

- ¥ Unbefugter Zugriff: Wenn ein:e Angreifer:in Zugriff auf das Repository erhält, kann er:sie leicht sensible Anmeldeinformationen abrufen. Dies könnte zu unbefugtem Zugriff auf Datenbanken, APIs und andere kritische Dienste führen, wenn die Geheimnisse noch gültig sind.
- ¥ Datenleck: Kompromittierte Anmeldeinformationen können zu Datenlecks führen, bei denen sensible Informationen abgerufen, geändert oder exfiltriert werden. Dies kann schwerwiegende rechtliche und finanzielle Folgen haben.

- ¥ Lateral Movement: Angreifer:innen mit Zugriff auf Klartext-Anmeldeinformationen können diese nutzen, um sich seitlich im Netzwerk zu bewegen und zusätzliche Systeme und Dienste zu kompromittieren.
- ¥ Ruf und Vertrauen: Die Anwesenheit von Geheimnissen im Repository kann das Vertrauen und den Ruf des Entwicklungsprozesses beeinträchtigen und den Stakeholdern einen Bedarf an verbesserten Sicherheitspraktiken anzeigen.

## Empfehlungen

- ¥ Entfernen Sie Geheimnisse aus der Git-Historie: Identifizieren und entfernen Sie alle Geheimnisse aus der Git-Repository-Historie. Dies kann mit Tools wie git filter-repo oder BFG Repo-Cleaner durchgeführt werden, um die Historie neu zu schreiben und sensible Informationen zu bereinigen.
- ¥ Drehen Sie exponierte Geheimnisse: Drehen Sie sofort alle exponierten Geheimnisse. Dies beinhaltet das Aktualisieren von Passwörtern, das Regenerieren von API-Schlüsseln und das Ersetzen von privaten Schlüsseln. Stellen Sie sicher, dass alle Systeme und Dienste, die die exponierten Geheimnisse verwenden, entsprechend aktualisiert werden.
- ¥ Verwenden Sie Umgebungsvariablen und Geheimnis-Verwaltungstools: Speichern Sie Geheimnisse in Umgebungsvariablen oder verwenden Sie Geheimnis-Verwaltungstools wie HashiCorp Vault, AWS Secrets Manager oder Azure Key Vault, um sensible Informationen sicher zu verwalten.
- ¥ Implementieren Sie Pre-Commit Hooks: Verwenden Sie Pre-Commit Hooks, um zu verhindern, dass Geheimnisse in Zukunft in das Repository committet werden. Tools wie pre-commit können mit Gitleaks oder ähnlichen Tools konfiguriert werden, um dies durchzusetzen.



## 5.13. Finding 24-4:13: Vulnerable Dependencies

The application uses some dependencies with known vulnerabilities.

(CVSS:3.1/AV:N/AC:H/PR:N/UI:R/S:U/C:H/I:H/A:H - HIGH 7.5)

### Prerequisites

The prerequisites for abusing a known vulnerability vary. In general, they are rather low as soon as suitable Proof of Concepts (PoCs) have been released.

### Description

The project uses poetry as a dependency management tool for python libraries. The dependencies are checked in the CI pipeline via pip-audit and a Renovatebot is configured for the repository, which creates merge requests to update dependencies that are regularly merged to the main branch. That is why the dependencies of the project are in general new and up to date. Still, running pip-audit manually on the latest commit of the test revealed the following vulnerable dependencies:

```
Found 3 known vulnerabilities in 3 packages
Name      Version ID                               Fix Versions
-----
idna      3.6      GHSA-jjg7-2v4v-x38h 3.7
jinja2    3.1.3    GHSA-h75v-3vvj-5mfj 3.1.4
werkzeug  3.0.2    GHSA-2g68-c3qc-8985 3.0.3
```

The highest known vulnerability of the werkzeug package was used to score this finding. It turned out that, while pip-audit is in fact run in the CI pipeline, the pipeline does not fail when pip-audit finds vulnerabilities. This is because the exit code is set to allowed in [line 126](https://git.example.com/dwh/DWHPY/-/blob/main/.gitlab-ci.yml?ref_type=heads#L126) [https://git.example.com/dwh/DWHPY/-/blob/main/.gitlab-ci.yml?ref\_type=heads#L126] of the CI file. A similar approach was found for poetry outdated.

Furthermore, the webpage uses an outdated version of jquery (3.4.1), found at the following URL:

[https://dwh-dev.example.com/\\_dash-component-suites/dash\\_admin\\_components/dash\\_admin\\_components.v0\\_1\\_4m1715689710.min.js](https://dwh-dev.example.com/_dash-component-suites/dash_admin_components/dash_admin_components.v0_1_4m1715689710.min.js)

This dependency originates from the dash-admin-components package with version 0.1.4 used in the DWH.

## Impact

- ¥ Abuse of security vulnerability: The impact of outdated package dependencies depends on the type of security vulnerability, but can be critical for the system that uses the dependency.

## Recommendations

- ¥ Direct instead of transitive dependencies: If updating the direct dependencies is not enough, the currently vulnerable python dependencies could be made a direct dependency in the poetry .toml file to enforce usage of the latest or a fixed version of the package. Due to dependency constraints, this might not always be possible, however.
- ¥ Making the CI Pipeline more strict: The CI pipeline should be changed not to ignore findings from pip-audit and poetry outdated anymore.
- ¥ Replacing unmaintained packages: Regarding the outdated jquery dependency, the package dash-admin-components which uses it has already the latest version 0.1.4. The last commit in the repository, however, is **about five years ago** [<https://github.com/dawidkopczyk/dash-admin-components>]. That is why it should be considered, if the functionality provided by the package could be replaced somehow.

## Further Information

- ¥ **National Vulnerability Library for CVEs at NIST** [<https://nvd.nist.gov/vuln>]
- ¥ **CVE for werkzeug 3.0.2** [<https://github.com/advisories/GHSA-2g68-c3qc-8985>]

## 5.14. Finding 24-4:14: Shared User Account on Host

The user account core on the DWH host machine is a shared user account with multiple authorized SSH keys assigned to different individuals.



Attack Vector	Local
Attack Complexity	Low
Privileges Required	Low
User Interaction	None
Scope	Unchanged
Confidentiality	None
Integrity	None
Availability	None

### Prerequisites

The attacker must be inside Example's internal network and authorized to be able to access the dwh machine via SSH.

### Description

There is only one user account on the host machine, which is shared by multiple individuals. This user account has multiple authorized Secure Shell (SSH) keys, meaning that several people use the same account for access. The user account is further used for running the dwh-app processes.

### Impact

- ¥ Accountability Issues: Shared accounts make it difficult to track individual actions, leading to a lack of accountability. It becomes challenging to determine who performed specific actions or changes on the system.
- ¥ Security Risks: If one of the authorized keys is compromised, the attacker gains access to the shared account and, consequently, to the host machine. This increases the potential attack surface and risk of unauthorized access.

## Recommendations

- ¥ Individual User Accounts: Create individual user accounts for each person who requires access to the host machine. This ensures that actions can be tracked and attributed to specific users. Ensure these user have only the necessary permissions.
- ¥ Dedicated User for DWH Processes: Create a user specifically for running web app processes. Ensure this user has only the necessary permissions to run the web app and remove any unnecessary access rights.

## 5.15. Finding 24-4:15: Passwordless Sudo

The user core can run all commands with root privileges without a password.



Attack Vector	Local
Attack Complexity	Low
Privileges Required	Low
User Interaction	None
Scope	Unchanged
Confidentiality	High
Integrity	High
Availability	High

### Prerequisites

The attacker must be inside Example's internal network and authorized to be able to access the dwh machine.

### Description

The user core is configured to run all commands with root privileges without being prompted for a password. This configuration is found in the sudoers file and allows the user core to execute any command as the root user without providing authentication.

The user core is the default user used for SSH login on the machine. This effectively grants root privileges to anyone who possesses a private key authorized for SSH access as the user core.



*Figure 17.1 sudoers file on the host machine. The user core and any member of the group sudo can use sudo without a password.*

## Impact

- ¥ Privilege Escalation: Any user or process that can assume the identity of core can escalate their privileges to root without any additional authentication. This greatly increases the risk of unauthorized access and potential system compromise.
- ¥ Non-Repudiation Issues: Actions performed using sudo without a password are harder to audit and track back to an authorized user. This complicates incident response and forensic investigations.

## Recommendations

- ¥ Require a Password for sudo: Configure the sudoers file to require a password for the core user and members of the group sudo when executing commands with sudo. This adds a layer of authentication and reduces the risk of unauthorized access.
- ¥ Review and Restrict sudo Permissions: Audit the sudo permissions granted to the core user and restrict them to only the necessary commands. Avoid granting blanket root access unless absolutely necessary.

## Further Information

- ¥ <https://www.flatcar.org/docs/latest/setup/security/hardening-guide/>

## 6. Appendix

### 6.1. EXAMPLE: Results npm audit - TEST

Last Commit: commit-hash

*Using node vX.Y.Z (npm vX.Y.Z)*

Keep additional files such as requests here.