Bundesamt
für Sicherheit in der
Informationstechnik

Deutschland
Digital•Sicher•BSI•

Results of the study on

static code analysis selected open source software CAOS 3.0

Vaultwarden

mgm security partners

results report
Security source code analysis and penetration testing

Project 604 – Code Analysis for Open Source Software (CAOS3)

Analysis results Work Package 4: "Vaultwarden"

June 12, 2024, version 1.1, status: final

mgm security partners GmbH
Taunusstraße 23
80807 Munich
Tel.: +49/89/358680-880
Email: info@mgm-sp.com
https://www.mgm-sp.com

A. Summary and evaluation

jump table

Table 1: Jump table for the most important areas in this document

The results of the security analysis are summarized on the following pages.

All references in this document are active links, meaning they can be accessed directly with a mouse click.

A1 Table of contents

A2 Management Summary

This report summarizes the results of the study of the application **Vaultwarden in version 1.30.3**and the**Bitwarden client browser extension in version 2024.3.1** together (see A4.1 for details)

The applications were subjected to a comprehensive tool-supported static source code analysis as well as an accompanying dynamic analysis (see A4.4). The investigation was carried out using the white box method, in which the testers had access to both the code and running instances.

The analyses were carried out in and on the contractor's premises between February 12, 2024 and May 16, 2024.

TheVaultwarden server application has two

security vulnerabilities of medium or high criticality

that an attacker can use to compromise users and the application.

Vaultwarden does not provide an offboarding process for members who leave the organization. This means that the master keys required for data access are not exchanged in this case. As a result, the departing member, whose access was actually revoked, still has the cryptographic key to the organization's data (E2.1). In combination with the vulnerability that encrypted data from other organizations can be accessed without authorization (E2.2), this former member therefore still retains unauthorized access to all secrets of the corresponding organization (including those created later) in plain text.

Furthermore, when changing the metadata of an emergency access that has been set up, the authorization of the accessing user is not checked (E1.1). The endpoint can be used to subsequently change the conditions of the emergency access that has been set up, including the access level and the waiting time. This would allow an attacker who has been granted access to an account in this way to access the data of the account with a higher access level and to arbitrarily shorten the waiting time set by the owner (7 days by default).

The admin dashboard is vulnerable to HTML injection attacks. By inserting HTML tags, it is possible to change the appearance and content of the page and, for example, embed links to malicious pages or, under certain circumstances, execute scripts (E3.1).

The comprehensive static source code analysis resulted in one SAST finding rated as [high] and 6 as [info], as well as one SCA finding rated as [high].
The SAST finding, which was classified as [high], was subsequently verified and confirmed by dynamic analysis (E3.1).

For theBitwarden client browser extension Apart from three SAST findings (A3.2) classified as [low] and one as [info], no vulnerabilities were identified.

A notice: A total of three CVE (Common Vulnerabilities and Exposure) entries were requested for the four above-mentioned security vulnerabilities of medium or high criticality. Due to the high correlation in the exploitability of the two vulnerabilities E2.1 and E2.2, a joint CVE entry was requested for the two findings.

At the time of completion of the report, no CVE numbers were available.

A3 overview of all findings

A3.1 Tool-based findings

Following the procedure for determining the hazard potential described in B1.3, this chapter summarizes all results of automatic tool analyses.

Notice : The SARIF exports differ in quantity from the findings in the Excel report, as we have excluded certain findings (see Section D for details) from an evaluation.

A3.2 SAST analyses

Based on the semi-automatic SAST and SECRETS analyses carried out (see Section D1), the following sets of findings result after mapping to our risk potential:

Vaultwarden Server:

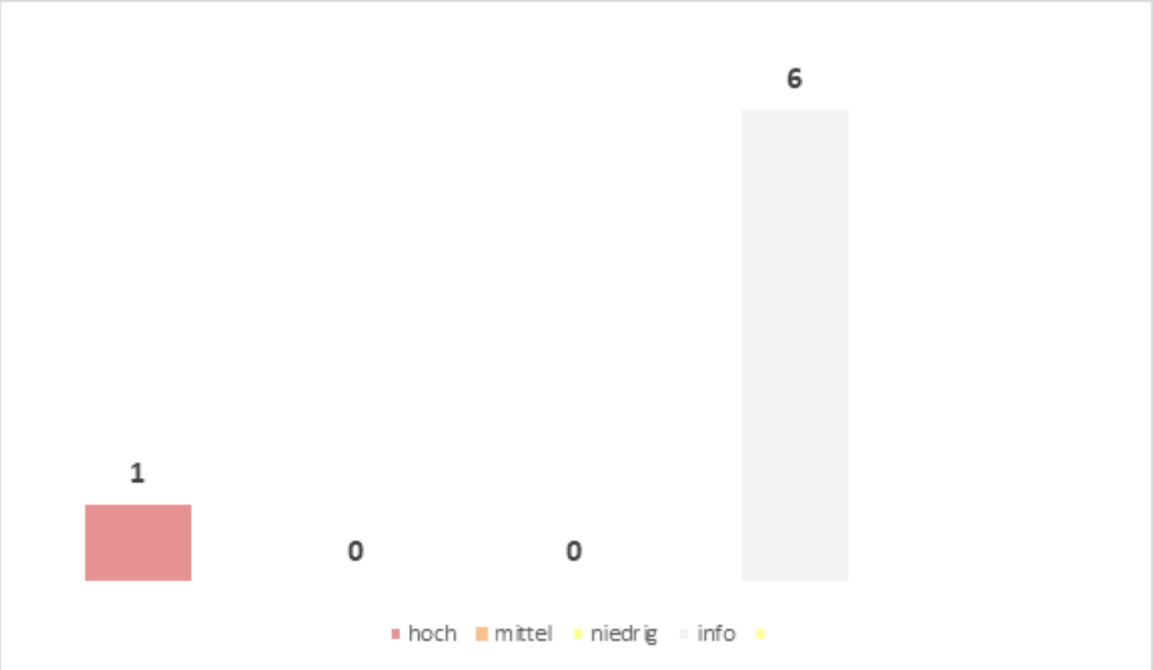


Figure 1: Overview of SAST findings Vaultwarden server

Bitwarden Client Browser Extension:

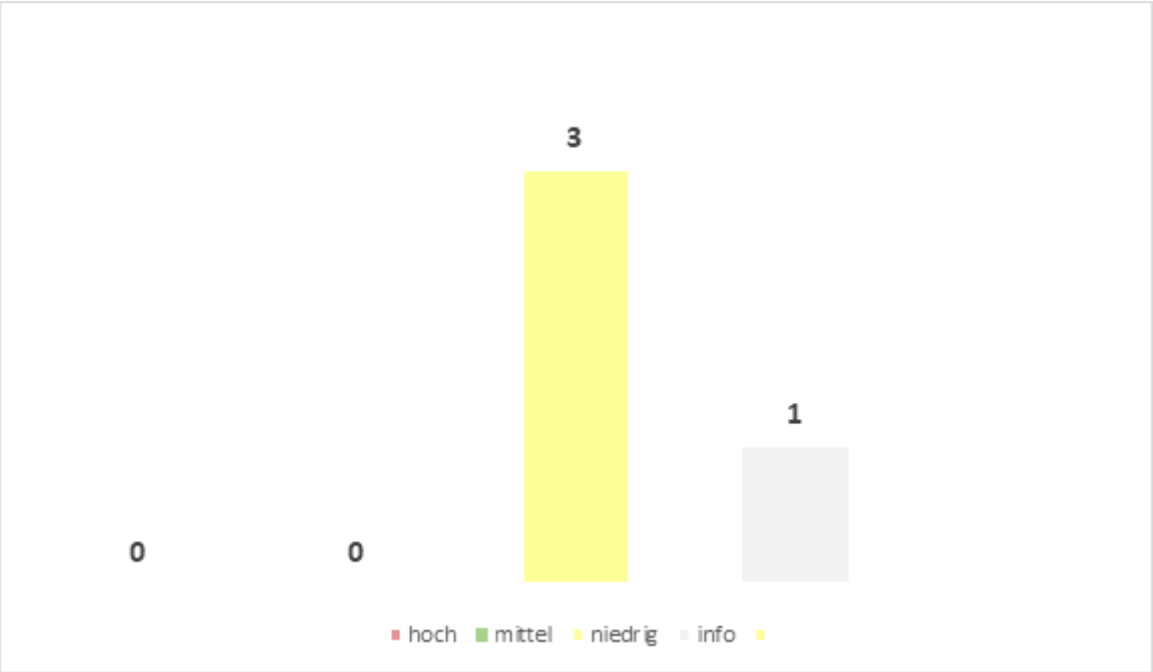


Figure 2: Overview of SAST findings Bitwarden client browser extension

A3.3 SCA analysis

Based on the automatic SCA analysis (see section D2), after mapping to our risk potential, the following sets of findings result for the Vaultwarden server:

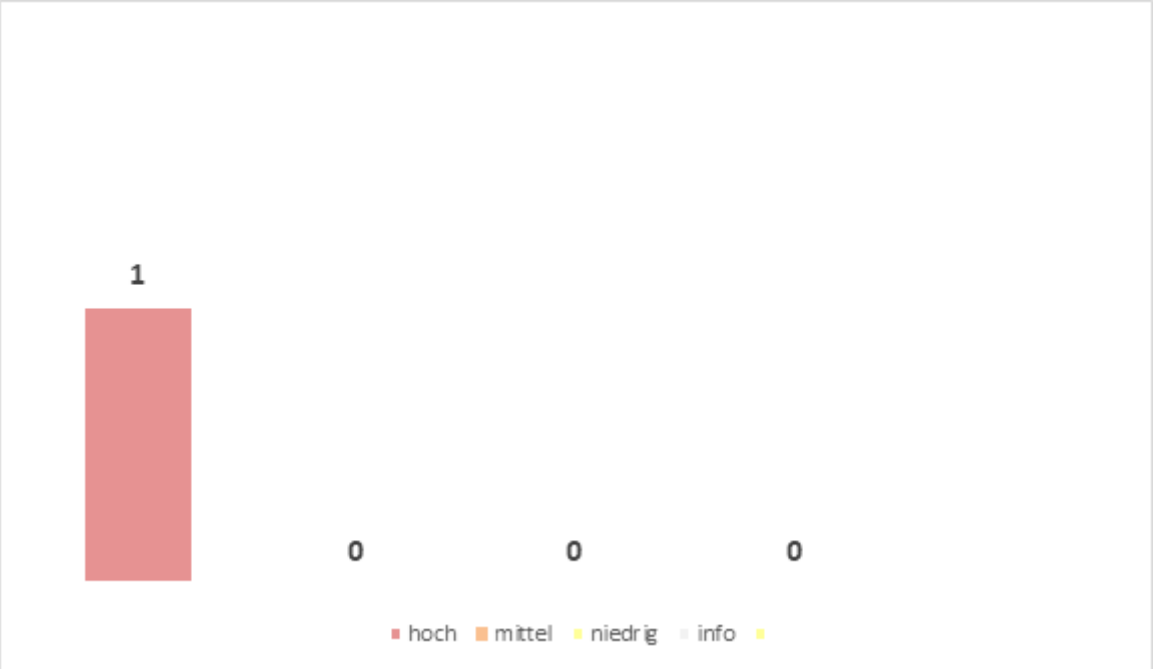


Figure 3: Overview of SCA Findings Vaultwarden Server

The SCA finding classified as [high] is not a direct but a transitive dependency.

Notice : No SCA analysis was performed for the Bitwarden Client browser extension because the external dependencies are managed and provided directly by the respective browser.

A3.4 Findings with detailed description in this document

The following overview lists the threats and vulnerabilities that were discovered using manual SAST or DAST approaches, or that we assessed as having a risk potential of “critical” or “high” in the automatic SAST analyses.

The following overview lists all vulnerabilities that are associated with low, medium, or high were assessed.

Table 2: Overview of detailed vulnerabilities

vulnerability	[low]	[medium]	[high]
authentication			
Unauthorized modification of the metadata of an Emergency Access			E1.1
access control			
Lack of rotation of organizational keys			E2.1
Unauthorized access to encrypted data		E2.2	
Possible hardening measures of the Docker environment	E2.3		
cross-site scripting			
HTML injection possible		E3.1	
Server-Side Request Forgery			
Server-side request forgery possible	E4.1		
Session ID in the URL or referrer			
Transmission of sensitive data in the URL	E5.1		
logic			
Denial-of-Service: IP-based user blocking possible	E6.1		
Insufficient anti-automation	E6.2		
disclosure of information			
disclosure of information	E7.1		
session management			
Insufficient cookie configuration in the admin dashboard	E8.1		
data validation			

vulnerability	[low]	[medium]	[high]
log injection	E9.1		

The following overview lists the observations that have the hazard potential [info].

Table 3: Overview of detailed info vulnerabilities

[info]	Chapter
Upload of any file format possible	E6.3

Overview

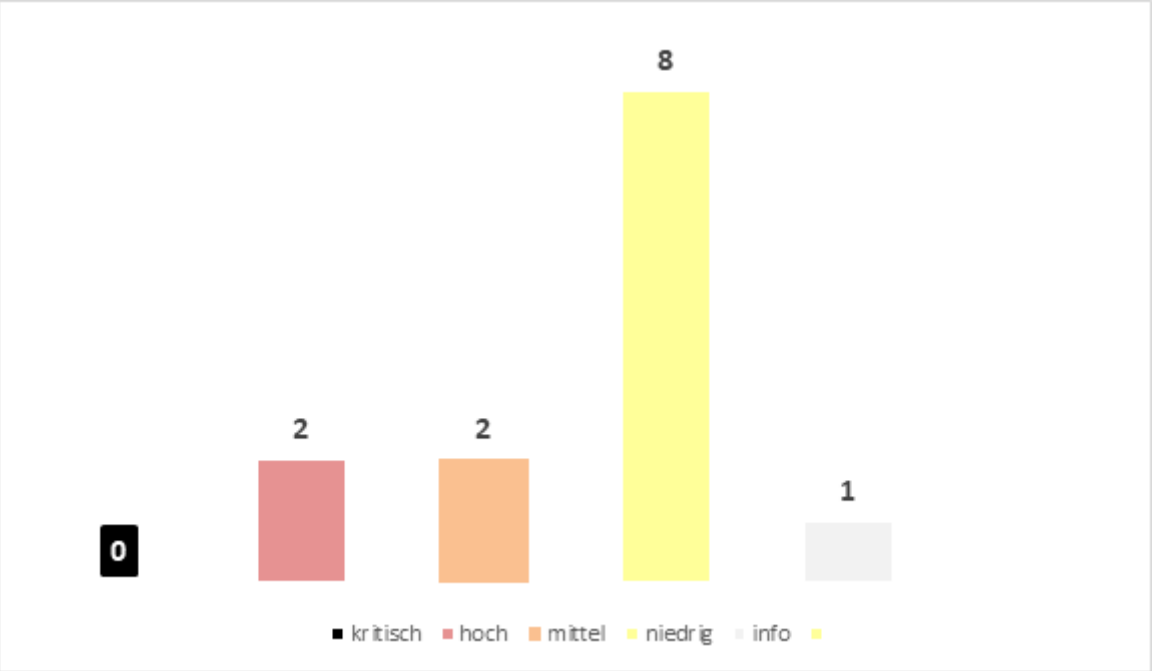


Figure 4: Statistics on the number of detailed findings in the document

A notice: No vulnerabilities were found in the Bitwarden Client browser extension.

A4 Assessment Details

A4.1 Tested Applications

In this security review, the following components were examined for vulnerabilities, both dedicated and dynamically in combination:

1. Vaultwarden Server (https://github.com/dani-garcia/vaultwarden) using the tagv1.30.3 (https://github.com/danigarcia/vaultwarden/releases/tag/1.30.3)
2. Bitwarden Client Browser Extension (subfolderapps/browserfrom https://github.com/bitwarden/clients) using the tagbrowserv2024.3.1 (https://github.com/bitwarden/clients/tree/browser-v2024.3.1)

The underlying quantity structures of the languages & technologies used (created with the tool cloc -https://github.com/AlDanial/cloc) are as follows:

github.com/AlDanial/cloc v 1.98 T=1.70 s (159.3 files/s, 46524.2 lines/s)

Language	files	blank	comment	code
Rust	52	3426	1692	21399
JavaScript	9	4704	9716	18895
CSS	4	917	48	11629
Handlebars	62	68	60	1394
SQL	107	145	47	1065
JSON	1	0	0	975
Text	1	117	0	544
YAML	10	70	85	506
SVG	6	1	1	425
Markdown	4	97	45	254
Bourne Shell	5	28	53	162
HCL	1	43	37	155
TOML	4	45	56	104
Python	2	17	20	75
INI	1	5	0	18
make	1	0	0	4
Dockerfile	1	0	0	1
SUM:	271	9683	11860	57605

Figure 5: Overview of the number of files & lines of code per technology used in the Vaultwarden server

```
github.com/AlDanial/cloc v 2.00  T=6.16 s (92.2 files/s, 43559.5 lines/s)
-------------------------------------------------------------------------------
Language                     files          blank        comment           code
-------------------------------------------------------------------------------
JSON                            70              0              0         187374
TypeScript                     319           7591           4528          45296
HTML                            84             68             46           7511
XML                             60           1327           2451           6418
SCSS                            14            564             82           3247
JavaScript                       7            108            242            789
Swift                            3             38              5            230
Markdown                         2             49              0            117
SVG                              6              0              0            110
CSS                              2              9              0             39
YAML                             1              0              0             28
-------------------------------------------------------------------------------
SUM:                           568           9754           7354         251159
-------------------------------------------------------------------------------
```

Figure 6: Overview of the number of files & lines of code per technology used in the browser extension

For the dynamic analysis parts, both components were installed on dedicated test systems. All (main) functionalities were manually triggered and examined by the tester (see also our methodology description in Section B), and all static findings classified as at least [high] were retested with regard to their actual exploitability.

A notice: The Bitwarden Web Client (https://github.com/bitwarden/clients/tree/webv2024.3.1) was not the focus of the tests, but was required for the penetration test of the server and the extension. Since important logic is also located here, it is recommended that this should also be subjected to more in-depth investigations in the future.

A4.2 test period

The tests were carried out between February 12, 2024 and May 16, 2024.

A4.3 Restrictive framework conditions

A4.3.1 source code analysis

Prior to the analysis, the following decisions were made regarding its implementation:

- no specific investigation of Vagrant/Docker releases + configurations

-no specific investigation of the source code of 3rd party dependencies

- no surveys/interviews of project participants (managers, developers, etc.)

- no investigation of mobile code components (Android, iOS code, etc.)

- no examination of demo or test code or documentation

- no evaluation of findings in development scripts and utilities that are not intended for productive use

-free choice of free and commercial analysis tools

A4.3.2 penetration test

Prior to the analysis, the following decisions were made regarding its implementation:

- Modules for dynamic analysis are determined by the contractor
- Installation of the applications by the contractor on its own infrastructure
- Vulnerability focus is determined by contractor (see also the specific threats listed in Chapter A4.5)

Notice : Results obtained by this approach can be found in Chapter E.

A4.4 Tool-supported analyses

Both applications were subjected to a thorough security investigation using a combination of the following approaches:

- Automatic and manual source code analysis (SAST + SECRETS + SCA)
- Automatic and manual penetration tests (DAST)

The specific tools used in each approach are listed in the following sections. All tool results are included in this report both in tool-specific RAW format and in report form (in Excel format) as an appendix (see also Section D).

All results reports are fully audited and all findings have been evaluated (our evaluation procedures are described in B1.3).

A coarse-grained overview of the findings initially provided by the tool (including their tool-specific criticality classification) as well as the assessments made by the auditor can be found in Section D.

Notice : Not all tools are able to comprehensively analyze all languages and frameworks. The tools were selected by the contractor, in each case appropriate to the programming languages and frameworks used.

A4.4.1 SAST: Tools used

The modules to be analyzed were scanned with the following SAST tools:

- Synopsys Coverity (https://www.synopsys.com/software-integrity/static-analysistools-sast/coverity.html) – standard rate, as of April 2024

  -Results see chapter D1.1

- Semgrep Professional (https://semgrep.dev/products/pro-engine/semgrep) – Standard rates, as of February 2024

  -Results see chapter D1.2

- Devskim (https://github.com/microsoft/DevSkim) – standard rate, as of April 2024

  -Results see Chapter D1.3

- Checkov (https://www.checkov.io/) – standard rate, as of April 2024

  -Results see Chapter D1.4

- Bearer (https://github.com/Bearer/bearer) - Standard rule rate, as of April 2024

  -Results see Chapter D1.5

- Snyk Code (https://snyk.io/de/) - Standard rule rate, as of April 2024

  -Results see chapter D1.6

- CodeQL (https://codeql.github.com/) - Standard rule rate, as of April 2024

  -Results see chapter D1.7

- Trivy (https://github.com/aquasecurity/trivy) - Standard rule rate, as of April 2024

  -Results see chapter D1.8

- Application Inspector (https://github.com/microsoft/ApplicationInspector) - Standard rule rate, as of April 2024

  -Results see chapter D1.9

A4.4.2 SECRETS: Tools used

A check for possible leaked secrets in the source code was carried out using the following tools:

-GitLeaks (https://gitleaks.io/) – standard rate, as of April 2024

-Results see chapter D1.10

- CredScan (as part of SAST-Scan -https://github.com/ShiftLeftSecurity/sastscan) – standard rate, as of April 2024

  -Results see chapter D1.11

A4.4.3 SCA: Tools used

The dependencies of the Vaultwarden server application were subjected to a known vulnerability check using the following SCA tools:

- Semgrep Supply Chain (https://semgrep.dev/products/semgrep-supply-chain/) – Known vulnerability database status as of March 15, 2024

  -Results see chapter D2.1

- Syft/Grype (https://github.com/anchore/syft ,https://github.com/anchore/grype) – Known vulnerability database status as of March 15, 2024

  -Results see chapter D2.2

A4.4.4 DAST: Tools used

For dynamic analyses, Burp Suite Professional was used (https://portswigger.net/burp/pro) in version v.2024.4.5. All (main) functionalities were manually triggered and examined by the tester. The "Burp Web Vulnerability Scanner" (active and passive) was used extensively.

All requests classified by the tester as "interesting from a security perspective" were analyzed in more depth manually (with automatic tool support). The following extensions were used (in addition to the built-in auxiliary tools such as Repeater or Intruder):

- Authorize v1.7
- Turbo Intruder v1.42
- Param Miner v1.4f
- Active Scan++ v1.0.24
- CSP-Bypass v1.0
- 403 Bypasser v1.2

The complete work protocols are stored in the file attachment.

The browser used for the dynamic server and extension tests was Google Chrome version 124 (with multiple updates during the test period).

Notice : Findings obtained which, according to our evaluations, may have led to valid findings/weaknesses are not explicitly listed in the appendix for this tool as a RAW format or report, but lead to finding descriptions in Chapter E.

A4.5 Manual analyses

As part of the investigations, extensive manual code reviews were also carried out, which were not possible in full due to the LoC (approx. 300,000 in total). Targeted investigations were therefore carried out at neuralgic points. The following prioritizations and focuses, initially defined together with the client, formed the basis for this:

A4.5.1 Vaultwarden Server

The Vaultwarden server itself does not contain any sensitive plaintext information, as all secrets stored on it are sufficiently encrypted according to the current state due to the end-to-end encryption. The main focus is therefore on the server and the cryptographic algorithms used for encryption.

-Specific threats:

- Browser-based threats and attacks (e.g. server-side request forgery, cross-site scripting ...)
- Server-based threats and attacks (e.g. command injection, SQL injection, ...)
- Incorrect access control (e.g. unauthorized access to unassigned *Collections* within a *Organization*, unauthorized access to user management functions (eg admin-only feature),...)
- Attacks on the concept of *Shared Access* within *Organization Vaults* and other important functions (e.g. *Sends*, *Emergency Access*)
- Attacks on 2-factor authentication (e.g. brute force attacks, bypasses, ...)
- Cryptographic misconfigurations (e.g. use of compared to BSI TR-02102-2 weak cryptographic algorithms or faulty initialization)
- Race conditions that can potentially lead to an exploitable error state

A4.5.2 Bitwarden client browser extension

The Bitwarden browser extension uses *auto-fill*-Function Cross-Origin Communication. The communication is with JavaScript *postMessage* Due to the high need for protection when transferring sensitive data to third parties, a great focus was placed on the secure authentication of the received *postMessage*- News.

-Other specific threats:

- Client-side vulnerabilities (e.g. cross-site scripting, DOM-based open redirects, faulty content security policy, clickjacking ...)
- Misconfiguration within the `manifest.json` (eg *permissions, host_permissions*, *web_accessible_resources*, ...)
- Disclosure of information in the code or memory of the browser extension

-Errors in the implementation of cryptographic functions

- Cryptographic misconfigurations
- Attacks on the process and business logic

A review of the CVEs was also carried out for both modules (starting on January 1, 2023). The primary focus was on analyzing and evaluating the way in which the reported vulnerabilities were handled and the quality of the correction provided by the project (see separate Chapter C).

Notice : When evaluating the findings identified with the automatic tools (see Chapter D), the auditor always looks at the "surrounding code" and can identify other weaknesses or so-called "code smells". In doing so, he learns how the individual developers program, what their preferences are and what errors could be "expected" from them. The evaluation of these findings is therefore always a form of "guided manual analysis".

B. Methodology and Evaluation

B1 Description of the methodology

B1.1 test procedure (white box test)

The application was subjected to both manual and automatic static and dynamic analyses.

B1.1.1 type of vulnerability assessment

We consider levels (aspects) 2 to 5 of web application security shown in the following image:

Table 4: Overview of the level model

	level	Content (Examples)
5	semantics	protection against deception and fraud - Information enables social engineering attacks - Use of pop-ups etc. facilitates phishing attacks - No protection in case the website is faked
4	logic	Securing processes and workflows as a whole - Using insecure email in an otherwise secure workflow - Password vulnerability due to poorly designed "forgot password" function - The use of secure passwords is not enforced
3	implementation rung	Avoiding programming errors that lead to vulnerabilities - Cross-Site Scripting - SQL injection - Cross-Site Request Forgery (Session Riding)
2	technology	Correct choice and safe use of methods - unencrypted transmission of sensitive data - Authentication procedures that are not appropriate to the protection requirements - Insufficient entropy of tokens

1	**system**	Securing the software used on the system platform - Error in the configuration of the web server - Known vulnerabilities in the software products used - Lack of access protection in the database
0	**Network & host**	securing host and network

Figure B1.1.1: Levels of Web Application Security

Level 0 – Network and Host (only partially subject of the investigation)

Web security also depends on network, hardware and host security. We look at the points of contact between the two levels where necessary.

Level 1 – System level (only partially subject of the investigation)

Level 1 includes all the software that a web application needs to run at all. This includes the web server and the application server, but also the database and the backend systems involved. All of these components must be taken into account when considering the security of a web application. A web application A that is programmed free of security flaws is still insecure if the database it uses can be manipulated via another channel, such as direct access via a database client that is not sufficiently secured and accessible to 'insiders', and this manipulation on the web can be exploited by an attacker.

A special field is the so-called known vulnerabilities, which we also assign to the system level.

Level 2 – Technology

This area is about selecting the right technology for the respective purpose and protection requirement - and using it correctly. A web application that transfers sensitive data unencrypted over the Internet is not using the right technology. And a web application that encrypts passwords but does so with a key that is too short is using the right technology incorrectly. The first application is not protected against spying during transmission, the other is not sufficiently protected against password cracking. Both are therefore insecure, even if they do not contain any security gaps in the program code.

Level 3 – Implementation

The implementation layer is the obvious layer of web application security. This is the area where unintentional programming errors (bugs) occur and lead to security problems, or faulty programming, such as missing or insufficient data validation, occurs.

Logic (Level 4) and Semantics (Level 5)

These two levels are the ones that are currently barely considered in the context of security. However, they are of great importance - and will become increasingly important in the age of phishing and identity theft - if the security of web applications is understood not only as protecting the server from intrusion attempts, but as a comprehensive concept. A web application is secure when it is secure together with the system that hosts it, and when it protects users and their trusted data from harm. The provider of a web application is therefore responsible not only for his own system, but also for everyone involved in using it. This last aspect is particularly important at the logic and semantics levels.

Level 4 – Logic

This level concerns the logic of the processes in an application - the application and business logic - with the user. If this is implemented too 'purpose-oriented', ie if the possibility that it will be used in a different way than intended is not taken into account enough, then it is often vulnerable. For example, are mechanisms built in that log the user out or provide other forms of protection in the event of improper use (conditions that are only possible by bypassing the browser functionality and thus clearly indicate misuse)?

Level 5 – Semantics

The semantic level of web application security includes aspects related to content and communication. This concerns the type of information that is given to the user, how content is presented to them and how it is handled. This area can rarely be limited to a single application. Rather, it is usually defined across websites or companies and must be followed by all applications, just as the CI specification must be adhered to by all communication media.

In particular, incorrect handling of the semantic aspects of web application security facilitates attacks on the user, which in turn damages the user's trust in the application and the company, and the web as a whole. Such attacks include social engineering, phishing, identity theft, deception, forgery, fraud, and breaching of data protection and privacy protection.

B1.2 evaluation scheme

B1.2.1 hazard potential cial

The tests carried out were each rated with a risk potential. The risk potential is an abstract measure for the classification of a vulnerability and the threat it poses.

The risk potential for each tested vulnerability is listed in the right column. We distinguish between:

[OK] the vulnerability mentioned does not exist

[critical] (see risk matrix below)

[high] (see risk matrix below)

[medium] (see risk matrix below)

[low] (see risk matrix below)

[info] this is information, not a vulnerability

The hazard potential is not a measure of the risk that such a vulnerability represents. The risk (= level of damage * probability of occurrence) that this hazard potential represents is not assessed here, as further information (e.g. damage potential) from the client is required to assess it.

B1.2.2 probability of occurrence

The probability of occurrence is essentially determined by the following factors:

- How easy is it to find the vulnerability (visibility)?
- How easy is the vulnerability to exploit?
- How well known is the vulnerability (Publicly Known)?
- What technical knowledge is required to exploit it?
- What access options are available (remote vs. local)?
- What is the vulnerability type?

To put it simply, the probability of occurrence depends on the difficulty of the attack.

The assessment of the probability of occurrence should only be used with great caution to decide whether a vulnerability needs to be eliminated. A suitably motivated attacker with sufficient knowledge is usually superior to a pentester if he is willing to invest enough time. The situation worsens further if one has to assume that a large number of attackers are looking for vulnerabilities.

B1.2.3 potential for damage

The potential for damage is primarily the consequence that can result from exploiting the vulnerability:

- loss of availability
- loss of confidentiality
- loss of completeness or integrity
- loss of protective mechanisms

-Expansion of access rights.

The specific potential damage can generally only be determined by the client.

risk/hazard matrix

It is common practice to calculate the risk using the formula

Risk = damage potential * probability of occurrence

to determine. From the descriptions of the probability of occurrence and the potential for damage, it is clear that both factors are also made up of several values. For example, a vulnerability that is generally known and for which exploits already exist has a high probability of occurrence. If this makes it possible, for example, to take over the system or to view confidential data, then the potential for damage is also high.

In order to calculate the product of probability of occurrence and damage potential with their own factors, there are various models (CVSS, DREAD, STRIDE, OCTAVE) that weight the individual factors differently.

In order to obtain an accurate risk assessment, the risk would have to be calculated using the client's subjective weighting according to the known models. This must be done individually for each vulnerability listed in this report.

In order to classify the vulnerabilities quickly and easily, the following simplified matrix is used for the risk potential:

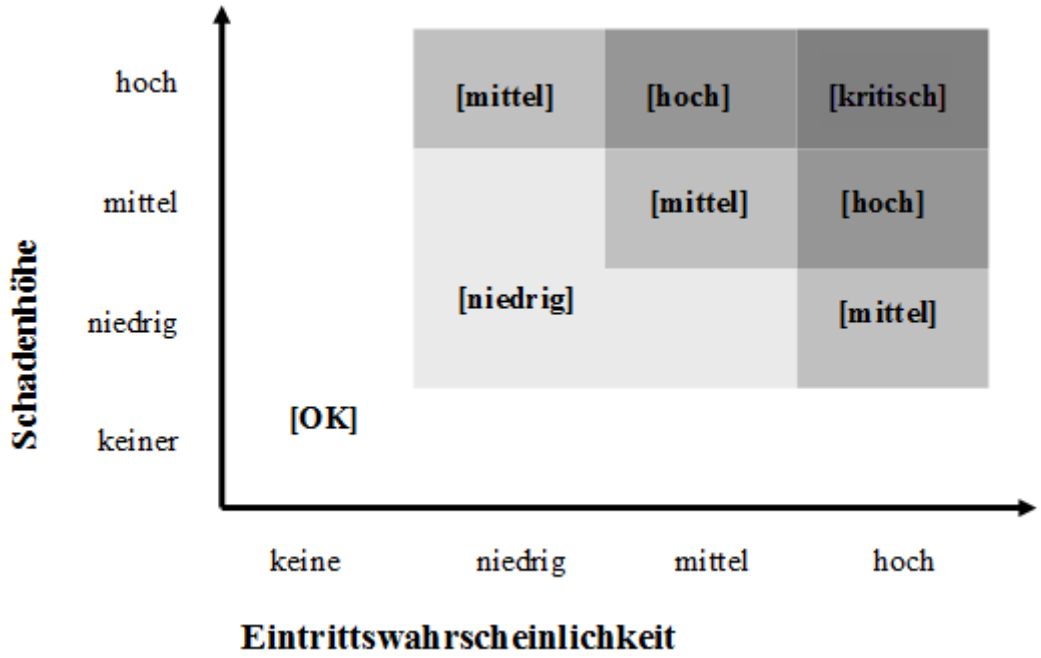


Figure 7: Determination of hazard potential

A priority for action can also be derived from this assessment:

[critical]–Immediate action is required; serious damage may be caused to the application, database or system; the system may need to be shut down

[high]–There is an urgent need for action, the web application/system and/or data are at risk;

[medium]–There is a need for action, the web application/system must be upgraded, e.g. by installing security updates or by additional
Security measures (e.g. WAF) must be used to protect against this.

[low]–The need for action is at the discretion of the client.

Notice

Although we based our assessment on objective criteria or generally accepted principles and tried to weigh up the advantages and disadvantages to the best of our knowledge and belief, such an assessment can only ever be subjective. Details unknown to us or different interpretations of the safety requirements can lead to a different assessment than the one given here. The client is therefore required to make their own assessment based on the information provided or to judge the urgency of correcting individual defects themselves.

We are happy to provide advice.

B1.3 Structure of the test results of the tool-based findings

All vulnerabilities reported by the tools were evaluated and result reports (if possible or necessary) are included in the appendix (see respective tool in section D or rough overview in section E).

Thetool's own criticality ratings were interpreted by us as follows:

- **Critical**(in the tools: "Critical")
- **High**(in the tools: “High” or “Error”)
- **Medium**(in the tools: "Medium", "Moderate" or "Warning")
- **Low**(in the tools: "Low", "Note" or "Recommendation")

The actual evaluations were carried out either in the interface provided by the tool or in Excel documents created by us from the RAW results. In some cases, the evaluation had to rely on the tool's own "wording". Thereviews in the reports are to be interpreted as follows:

-**Confirmed**(in the ratings: "Exploitable" or "Confirmed")

- Finding was confirmed (findings rated “critical” or “high” were examined for their exploitability and then described in more detail in Section E)

-**Suspicious**(in the reviews: "Suspicious")

- Finding could not be 100% confirmed or refuted (neither statically nor by DAST analysis), but could represent a (future) security issue that the project should be aware of. Here, the assessment and treatment must be decided or completed by a project insider with a security background.

-**Bad Practice**(in the reviews: "Bad Practice")

- Finding was classified as a programming error or poor code quality, which may result in security-related consequences or logical misinterpretationscould However, the probability of this happening was considered very low.

-**No problem**(in the reviews: "Not an issue")

-Finding was classified as irrelevant or false positive.

Based on these two classifications, the following interpretation of the risk/hazard matrix defined in 0 results (the statistical overview in A3.1 is based on this assignment):

[critical]

-Tool classification:critical X auditor ratingconfirmed

[high]

-Tool classification:high X auditor ratingconfirmed

[medium]

- Tool classification:critical X auditor ratingsuspicious
- Tool classification:medium X auditor ratingconfirmed

[low]

- Tool classification:critical X auditor ratingbad practice
- Tool classification:high X auditor ratingsuspicious
- Tool classification:low X auditor ratingconfirmed

[info]

- Tool classification:high X auditor ratingbad practice
- Tool classification:medium X auditor ratingsuspicious
- Tool classification:medium X auditor ratingbad practice
- Tool classification:low X auditor ratingsuspicious
- Tool classification:low X auditor ratingbad practice

B1.4 Structure of the test results in this document (manual SAST/DAST)

For each relevant vulnerability, there is a separate subchapter, with headings providing information about the type of vulnerability.

Such a heading might look like this:

- nm Cross-Site-Scripting
- **nm** is the subchapter number
- **cross-site scripting** is the exact vulnerability name

Each of these subchapters is then divided into:

- **threat**
  The vulnerability or attack technique is explained in a way that is easy to understand. If necessary, with links to further sources of information.
- **observation**
  If the application is immune to the described threat, this will be justified and proven here.

  If the application is vulnerable to the described threat, the observation is explained and documented in a separate subsection as follows.

nmo reflected cross-site scripting widely used [high]

- **Reflected cross-site scripting widely used**
  is a brief summary of the vulnerability found
- **high**
  is the assessment of the hazard potential

The structure looks like this:

- **observation**
  The test and observation of the application's response are described. The aim is to (1) enable the reader to understand the attack in order to better assess the risk themselves and (2) to make the attack understandable. It is not always possible to guarantee (2), as replication may require more extensive preparation or temporal correlation with other conditions.
- **reproductive steps**(optional)
  If the specific exploitation of the vulnerability requires more detailed instructions that cannot reasonably be covered by observation, the individual steps for reproducing the vulnerability are described here.
- **vulnerability in the code**(optional)
  This section refers to the line(s) in the code or the code sections that are responsible for the vulnerability shown. If necessary, individual elements of the code are briefly explained. The section is only

present if the vulnerability occurred in the source
code of the object under investigation.

- **exploitability**
  This describes the conditions that must be met for an attack, how an attacker can exploit the vulnerability and what consequences and risks are to be expected. In general, we use a worst-case approach, ie when in doubt, we describe the more threatening scenario. This description is independent of the probability of occurrence. An assessment of the probability of occurrence must be made separately.

- **Note / Remark**(optional)
  Here, special features and/or specific conditions of the vulnerability found are described, e.g. if it did not occur reproducibly or only with certain browsers. Such comments are often important for assessing the rating (see above).

- **measure**
  A distinction must be made between basic measures and individual measures. Basic measures are usually threat-related, whereas individual measures are more vulnerability-related. Basic measures can be formulated independently of the respective application and are universally valid.

  For web applications that have been developed without taking basic protection measures into account, the subsequent application of basic measures can lead to an unacceptably high level of effort. In addition, an application that has not implemented the described measure is not necessarily vulnerable to an attack because this has been deliberately ensured by other measures or because it is more of a coincidence. In this case, it would not be appropriate to require the implementation of the measure mentioned.

  Measures from the perspective of penetration testing are therefore mostly individual vulnerability-related measures that only become relevant when a vulnerability is discovered. Often, there is not just one measure that can be considered to eliminate the vulnerability, but a whole range of possible measures. The decision on the right measure must be made individually for each application and depends on the software-related constraints, the technology used, the type of implementation, the implementation costs, etc.

- An actual assessment and decision on how to eliminate a vulnerability must be made by the client or it will be discussed in a final workshop together with the developers or service providers.
  application managers.

C. CVE reviews

As part of this analysis, all reported CVEs (01/01/2023 to 05/16/2024) were reviewed and commented on for their correction.

During the above period, a CVE was reported for the browser extension:
CVE-2023-27974

C1 CVE-2023-27974

Table 5: Profile of the CVE entry

fixed assessment	OK – we support the manufacturer’s position.
CWE	unclassified
CVE short description	Bitwarden through 2023.2.1 offers password auto-fill when the second-level domain matches, eg, a password stored for an example.com hosting provider when customer-website.example.com is visited. NOTE: the vendor's position is that "Auto-fill on page load" is not enabled by default.
CVE registered	n/a
CVE published	March 8, 2023
CVE details	https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2023-27974 https://www.cve.org/CVERecord?id=CVE-2023-27974 https://nvd.nist.gov/vuln/detail/CVE-2023-27974
CVSS base score (NIST)	n/a
Summary	An attacker could use a subdomain of a known/trusted domain (e.g.evil.example.com, assumingexample.comwould be trusted) to pass passwords through theauto-fillfeature to steal. If a malicious form were embedded via an iframe in a legitimate page with a form, theauto-fillFeature fill out both forms.
Affected versions	<= 2023.2.1
fixed date	n/a
fix commit	n/a

C1.1 correction information

Has not been corrected yet.

C1.2 Evaluation

The risk of the reported security issue is low, as a successful attack against Bitwarden users requires a maliciousiframeon a legitimate login page.

If the login page also contains common restrictions foriframesThis would also significantly mitigate the problem.

D. Test results automatic tools

The results of the study are summarized below, divided according to the automatic analysis tool. Detailed results can be found in the results reports and documents in the appendix.

After performing the scan, findings in the following folders were excluded from review and evaluation:

- . github/ (Github working files)
- . vscode (Visual Studio working files)
- spec/ (tests and fixtures)

Notice : The finding raw material (SARIF or JSON files) may also contain results for the above folders.

D1 SAST and SECRET tools

All findings evaluated and commented on by us can be seen in the following Excel result documents:

- *vaultwarden-server-audited.xlsx*
- *vaultwarden-browser-extension-audited.xlsx*

Findings excluded from the review and identified as duplicates are documented in the following separate Excel files (in the following descriptions, affected findings are generally referred to as “excluded from the evaluation”):

- *vaultwarden-server-excluded.xlsx*
- *vaultwarden-browser-extension-excluded.xlsx*

D1.1 Synopsys Coverity

These scans were performed using the standard rules (as of April 2024). The scanner engine used was version 2023.12.0.

The initial finding classification was based on the Coverity standard rating (high, medium, low). The findings provided by the tool that were not excluded by us were transferred to the central Excel results document for the audit, in which the rating was carried out.

The findings from the scan are briefly summarized below.

D1.1.1 Supplied report artifacts

The raw (unaudited) data provided by the tool can be found in the following files:

- *raw/vaultwarden-server-coverity-raw.json*
- *raw/vaultwarden-browser-extension-coverity-raw.json*

D1.1.2 Rough overview of the raw scan result of the tool

The following lists the number of findings to be assessed (per standard finding classification), as well as in brackets the actual number of findings found or excluded:

Vaultwarden Server

- 1x "medium"
- 3x "low"

browser extension

- 29x “medium” (32 found and 3 excluded from the rating)
- 78x “low” (80 found and 2 excluded from the evaluation)

D1.1.3 Overview of the auditor's assessment

Summary of the auditor’s assessment (see Excel results document in the appendix for details):

Vaultwarden Server

- 1 rated “medium” finding resulted in the following classification:
  - -1 wasNo problemclassified
- 3 assessed “low” findings resulted in the following classifications:
  - 2 wereBad Practiceclassified
  - 1 wasNo problemclassified

browser extension

- 29 evaluated “medium” findings resulted in the following classifications:
  - -29 wereNo problemclassified
- 78 assessed “low” findings resulted in the following classifications:
  - -78 wereNo problemclassified

D1.2 Semgrep Professional

The following rule sets (as of May 2024) were used for these scans:

- p/owasp-top-ten
- p/cwe-top-25
- p/default
- p/comment
- p/r2c-security-audit
- p/command-injection
- p/secrets
- p/sql-injection
- p/xss
- p/bandit
- p/eslint
- p/brakeman
- p/security-code-scan
- p/security-audit
- p/supply-chain
- p/findsecbugs
- p/gosec
- p/jwt

The scanner engine used was version 1.67.0.

The initial finding classification was based on the Semgrep standard assessment (error, warning, note). The findings provided by the tool that were not excluded by us were transferred to the central Excel results document for auditing, in which the assessment was carried out.

The findings from the scan are briefly summarized below.

D1.2.1 Supplied report artifacts

The raw (unaudited) data provided by the tool can be found in the following files:

- *raw/vaultwarden-server-semgrep-pro-raw.sarif*

- *raw/vaultwarden-browser-extension-semgrep-pro-raw.sarif*

D1.2.2 Rough overview of the raw scan result of the tool

The following lists the number of findings to be assessed (per standard finding classification), as well as in brackets the actual number of findings found or excluded:

Vaultwarden Server

- 3x “error” (9 found and 6 excluded from the evaluation)

- 141x “warning” (416 found and 275 excluded from the evaluation)

browser extension

- 8x "error"

- 209x “warning” (224 found and 15 excluded from the evaluation)

D1.2.3 Overview of the auditor's assessment

Summary of the auditor’s assessment (see Excel results document in the appendix for details):

Vaultwarden Server

- 3 evaluated “error” findings resulted in the following classifications:

    - 1 wasConfirmedclassified

    - 2 wereNo problemclassified

- 141 evaluated “warning” findings resulted in the following classifications:

    -141 wereNo problemclassified

browser extension

- 8 evaluated “error” findings resulted in the following classifications:

    -8 wereNo problemclassified

- 209 evaluated “warning” findings resulted in the following classifications:

    -209 wereNo problemclassified

D1.3 Devskim

These scans were performed using the standard rules (as of April 2024). The scanner engine used was version 1.0.20 or 1.0.32 (browser extension).

The initial finding classification was based on the Devskim standard assessment (error, warning, note). The findings provided by the tool that were not excluded by us were transferred to the central Excel results document for auditing, in which the assessment was carried out.

The findings from the scan are briefly summarized below.

D1.3.1 Supplied report artifacts

The raw (unaudited) data provided by the tool can be found in the following files:

- *raw/vaultwarden-server-devskim-raw.sarif*
- *raw/vaultwarden-browser-extension-devskim-raw.sarif*

D1.3.2 Rough overview of the raw scan result of the tool

The following lists the number of findings to be assessed (per standard finding classification), as well as in brackets the actual number of findings found or excluded:

Vaultwarden Server

- 5x "error" (6 found and 1 excluded from the evaluation)
- 0x "warning" (2 found and 2 excluded from the evaluation)
- 44x “note” (57 found and 13 excluded from the evaluation)

browser extension

- 2x "error"
- 11x "warning"
- 354x "note"

D1.3.3 Overview of the auditor's assessment

Summary of the auditor’s assessment (see Excel results document in the appendix for details):

Vaultwarden Server

-5 evaluated “error” findings resulted in the following classifications:

-5 wereNo problemclassified

- 44 assessed “note” findings resulted in the following classifications:

-44 wereNo problemclassified

browser extension

- 2 evaluated “error” findings resulted in the following classifications:

-2 wereNo problemclassified

- 11 evaluated “warning” findings resulted in the following classifications:

-11 wereNo problemclassified

- 354 assessed “note” findings resulted in the following classifications:

-354 wereNo problemclassified

D1.4 Checkov

These scans were performed using the standard rules (as of April 2024). The scanner engine used was version 3.2.22.

The initial finding classification was based on the Checkov standard assessment (error, warning, note). The findings provided by the tool that were not excluded by us were transferred to the central Excel results document for the audit, in which the assessment was carried out.

The findings from the scan are briefly summarized below.

D1.4.1 Supplied report artifacts

The raw (unaudited) data provided by the tool can be found in the following files:

- *raw/vaultwarden-server-checkov-raw.sarif*

- *raw/vaultwarden-browser-extension-checkov-raw.sarif*

Rough overview of the raw scan result of the tool

The following lists the number of findings to be assessed (per standard finding classification), as well as in brackets the actual number of findings found or excluded:

Vaultwarden Server

-6x “error” (11 found and 5 excluded from the evaluation)

browser extension

-1x "error"

D1.4.2 Overview of the auditor's assessment

Summary of the auditor’s assessment (see Excel results document in the appendix for details):

Vaultwarden Server

-6 evaluated “error” findings resulted in the following classifications:

- 4 wereBad Practiceclassified
- 2 wereNo problemclassified

browser extension

-1 rated “error” finding resulted in the following classification:

-1 wasNo problemclassified

D1.5 Bearer

These scans were performed using the standard rules (as of April 2024). The scanner engine used was version 1.27.1.

The initial finding classification was based on the Bearer Standard assessment (error, warning, note). The findings provided by the tool that were not excluded by us were transferred to the central Excel results document for auditing, in which the assessment was carried out.

The findings from the scan are briefly summarized below.

D1.5.1 Supplied report artifacts

The raw (unaudited) data provided by the tool can be found in the following files:

- *raw/vaultwarden-server-bearer-raw.sarif*
- *raw/vaultwarden-browser-extension-bearer-raw.sarif*

D1.5.2 Rough overview of the raw scan result of the tool

The following lists the number of findings to be assessed (per standard finding classification), as well as in brackets the actual number of findings found or excluded:

Vaultwarden Server

-2x "error" (3 found and 1 excluded from the evaluation)

browser extension

-33x "error"

D1.5.3 Overview of the auditor's assessment

Summary of the auditor’s assessment (see Excel results document in the appendix for details):

Vaultwarden Server

-2 evaluated “error” findings resulted in the following classifications:

-2 wereNo problemclassified

browser extension

-33 evaluated “error” findings resulted in the following classifications:

-33 wereNo problemclassified

D1.6 Snyk Code

These scans were performed using the standard rules (as of April 2024). The scanner engine used was version 1.1284.0.

The initial finding classification was based on the Snyk Code standard assessment (error, warning, note). The findings provided by the tool that were not excluded by us were transferred to the central Excel results document for auditing, in which the assessment was carried out.

The findings from the scan are briefly summarized below.

D1.6.1 Supplied report artifacts

The raw (unaudited) data provided by the tool can be found in the following files:

- *raw/vaultwarden-server-snyk-code-raw.sarif*
- *raw/vaultwarden-browser-extension-snyk-code-raw.sarif*

D1.6.2 Rough overview of the raw scan result of the tool

The following lists the number of findings to be assessed (per standard finding classification), as well as the actual number of findings found or excluded in brackets:

Vaultwarden Server

-2x "warning"

-0x "note" (2 found and 2 excluded from the evaluation)

browser extension

-9x “warning” (14 found and 5 excluded from the evaluation)

D1.6.3 Overview of the auditor's assessment

Summary of the auditor’s assessment (see Excel results document in the appendix for details):

Vaultwarden Server

-2 assessed “warning” findings resulted in the following classifications:

-2 wereNo problemclassified

browser extension

-9 evaluated “warning” findings resulted in the following classifications:

-9 wereNo problemclassified

D1.7 CodeQL

These scans were performed using the standard rules (as of April 2024). The scanner engine used was version 2.15.1.

The initial finding classification was based on the CodeQL standard assessment (error, warning, note). The findings provided by the tool that were not excluded by us were transferred to the central Excel results document for auditing, in which the assessment was carried out.

The findings from the scan are briefly summarized below.

D1.7.1 Supplied report artifacts

The raw (unaudited) data provided by the tool can be found in the following files:

- *raw/vaultwarden-server-codeql-js-raw.sarif*
- *raw/vaultwarden-browser-extension-codeql-js-raw.sarif*

D1.7.2 Rough overview of the raw scan result of the tool

The following lists the number of findings to be assessed (per standard finding classification), as well as in brackets the actual number of findings found or excluded:

Vaultwarden Server

-0x "warning" (13 found and 13 excluded from the evaluation)

browser extension

-1x “warning” (2 found and 1 excluded from the evaluation)

D1.7.3 Overview of the auditor's assessment

Summary of the auditor’s assessment (see Excel results document in the appendix for details):

Vaultwarden Server

-No evaluation is made

browser extension

-1 rated “warning” finding resulted in the following classification:

-1 were consideredNo problemclassified

D1.8 Trivy

These scans were performed using the standard rules (as of April 2024). The scanner engine used was version 0.49.1.

The initial finding classification was based on the Trivy standard assessment (error, warning, note). The findings provided by the tool that were not excluded by us were transferred to the central Excel results document for auditing, in which the assessment was carried out.

The findings from the scan are briefly summarized below.

D1.8.1 Supplied report artifacts

The raw (unaudited) data provided by the tool can be found in the following files:

- *raw/vaultwarden-server-trivy-raw.sarif*
- *raw/vaultwarden-browser-extension-trivy-raw.sarif*

D1.8.2 Rough overview of the raw scan result of the tool

The number of findings (per standard finding classification) is listed below, as well as the actual number of findings found or excluded in brackets:

Vaultwarden Server

-**0x "error" (2 found and 2 excluded from the evaluation)**

-1x "warning"

browser extension

- 3x "error"
- 3x “warning” (4 found and 1 excluded from the evaluation)
- 3x "note"

D1.8.3 Overview of the auditor's assessment

Summary of the auditor’s assessment (see Excel results document in the appendix for details):

Vaultwarden Server

-1 rated “warning” finding resulted in the following classification:

-1 wasNo problemclassified

browser extension

- 3 evaluated “error” findings resulted in the following classifications:
  - 1 wasBad Practiceclassified
  - 2 wereNo problemclassified
- 3 assessed “warning” findings resulted in the following classifications:

-3 wereNo problemclassified

- 3 assessed “note” findings resulted in the following classifications:

-3 wereNo problemclassified

D1.9 Application Inspector

These scans were performed using the standard rules (as of April 2024). The scanner engine was used in versions 1.9.10 for the Vaultwarden server and 1.9.22 for the browser extension.

The initial finding classification was based on the Application Inspector standard assessment (error, warning, note). The findings provided by the tool that were not excluded by us were transferred to the central Excel results document for auditing, in which the assessment was carried out.

The findings from the scan are briefly summarized below.

D1.9.1 Supplied report artifacts

The raw (unaudited) data provided by the tool can be found in the following files:

- *raw/vaultwarden-server-appinspector-raw.sarif*
- *raw/vaultwarden-browser-extension-appinspector-raw.sarif*

D1.9.2 Rough overview of the raw scan result of the tool

The following lists the number of findings to be assessed (per standard finding classification), as well as the actual number of findings found or excluded in brackets:

Vaultwarden Server

-5,151x “note” (6,773 found and 1,622 excluded from the evaluation)

browser extension

-7,702x “note” (11,729 found and 4,027 excluded from the evaluation)

D1.9.3 Overview of the auditor's assessment

Summary of the auditor’s assessment (see Excel results document in the appendix for details):

Vaultwarden Server

-5,151 assessed “note” findings resulted in the following classifications:

-5,151 wereNo problemclassified

browser extension

-7,702 assessed “note” findings resulted in the following classifications:

-7,702 wereNo problemclassified

D1.10 GitLeaks

These scans were performed using the standard rules (as of April 2024). The scanner engine used was version 8.18.0.

GitLeaks does not initially classify findings in terms of criticality. All findings provided by the tool that were not excluded by us were transferred equally to the central Excel results document for auditing, in which the assessment was carried out.

The findings from the scan are briefly summarized below.

Notice : We enriched the exported SARIF files with the corresponding lines of code, with 5 lines of context before and after, to make it easier to use the results (reason: findings often refer to commits from the past).

This took place inresults-Array for the individual result entries under the JSON pathlocations > physicalLocation > artifactLocation > description > text.Schema conformity with regard to the Sarif format in version 2.1.0 was maintained.

D1.10.1 Supplied report artifacts

The raw (unaudited) data provided by the tool can be found in the following files:

- *raw/vaultwarden-server-gitleaks-raw.sarif*
- *raw/vaultwarden-browser-extension-gitleaks-raw.sarif*

D1.10.2 Rough overview of the raw scan result of the tool

The following lists the number of findings to be evaluated, as well as the actual number of findings found or excluded in brackets (note: Gitleaks does not perform a criticality rating):

Vaultwarden Server

-1x “warning” (classification was made by auditor)

browser extension

- 729x “warning” (929 found and 200 excluded from the assessment) (classification was done by auditor)

D1.10.3 Overview of the auditor's assessment

Summary of the auditor’s assessment (see Excel results document in the appendix for details):

Vaultwarden Server

-1 rated “warning” finding resulted in the following classification:

-1 were consideredNo problemclassified

browser extension

-729 evaluated “warning” findings resulted in the following classifications:

- 3 wereSuspiciousclassified
- 726 wereNo problemclassified

D1.11 CredScan

These scans were performed using the standard rules (as of April 2024). The scanner engine used was version 1.0.0-scan (as part of SAST-Scan version 2.1.2).

The initial finding classification was based on the CredScan standard assessment (error, warning, note). The findings provided by the tool that were not excluded by us were transferred to the central Excel results document for the audit, in which the assessment was carried out.

The findings from the scan are briefly summarized below.

D1.11.1 Supplied report artifacts

The raw (unaudited) data provided by the tool can be found in the following files:

- *raw/vaultwarden-server-credscan-raw.sarif*
- *raw/vaultwarden-browser-extension-credscan-raw.sarif*

D1.11.2 Rough overview of the raw scan result of the tool

The following lists the number of findings to be assessed (per standard finding classification), as well as in brackets the actual number of findings found or excluded:

Vaultwarden Server

-1x "error"

browser extension

-2x "error"

D1.11.3 Overview of the auditor's assessment

Summary of the auditor’s assessment (see Excel results document in the appendix for details):

Vaultwarden Server

-1 rated “error” finding resulted in the following classification:

-1 were consideredNo problemclassified

browser extension

-2 evaluated “error” findings resulted in the following classifications:

-2 wereNo problemclassified

D2 SCA tools

All findings found (and cleaned of duplicates) can be seen in the following Excel result documents:

-*vaultwarden-server-sca.xlsx*

Findings identified as duplicates are documented in the following separate Excel files:

-*vaultwarden-server-sca-excluded.xlsx*

Notice : Since the information obtained by the tools is based on well-known and widely used databases, we assume all findings to be "confirmed". The paragraph "Overview of the auditor's assessment" is therefore omitted for this tool category.

In our experience, a fine-grained evaluation (e.g. by not using vulnerable library parts) can only be carried out by project insiders.

D2.1 Semgrep Supply Chain

The following rule sets (as of March 2024) were used for these scans:

-p/supply-chain

The scanner engine used was version 1.67.0.

The initial finding classification was based on the Semgrep standard assessment (error, warning, note). All results were included in the SCA results document.

The findings from the scan are briefly summarized below.

D2.1.1 Supplied report artifacts

The raw (unaudited) data provided by the tool can be found in the following files:

-*raw/vaultwarden-server-semgrep-supply-chain-raw.sarif*

D2.1.2 Rough overview of the raw scan result of the tool

The number of findings (per standard finding classification) is listed below, as well as the actual number of findings found or excluded in brackets:

Vaultwarden Server

- 1 with tool classification “warning” (upgraded to “error” – see https://github.com/advisories/GHSA-r8w9-5wcg-vfj7)

browser extension

Not applicable.

D2.2 Syft / Grype

These scans were performed using the standard rules (as of March 2024). The scanner engine used was version 0.89.0 (Syft) or 0.66.0 (Grype).

The initial finding classification was carried out without a standard evaluation. Results that provided new insights compared to “Semgrep Supply Chain” were incorporated into the SCA results document.

The findings from the scan are briefly summarized below.

D2.2.1 Supplied report artifacts

The (unaudited) raw data provided by the tool can be found in the following file:

-raw/vaultwarden_server_grype_raw.sarif

D2.2.2 Rough overview of the raw scan result of the tool

The number of findings (per standard finding classification) is listed below, as well as the actual number of findings found or excluded in brackets:

Vaultwarden Server

- 0 with tool rating “error” (1 found and 1 excluded from the evaluation)

browser extension

Not applicable.

E. Test results DAST analyses detailed

This chapter lists observations that were discovered by manual DAST or described in detail as a result of retests of high or critical SAST findings.

E1 authentication

threat

Insufficient or faulty authentication remains a widespread vulnerability in web applications.

Authentication refers to the confirmation of the identity of an entity, e.g. a person or a device. Knowledge of information, ownership of an object or biometric characteristics can be used for this. Knowledge-based authentication using a user name and password is still the most common form of authentication. In order to increase the security of authentication, however, several characteristics are increasingly being combined, which is known as multi-factor authentication (MFA).

During authentication, only the identity is checked, but there is no check as to whether access is permitted. This check must be done via authorization.

E1.1 Unauthorized modification of the metadata of an Emergency Access [high]

observation

Emergency Access is a feature of Vaultwarden that allows a user (A*access grantor*) allows a trusted contact (G*rantee*) who can request access to his Vault in an emergency.

When setting up a *Emergency Access* can the A*access grantor* set some conditions for the access granted, including:

- *Access Level*: Determines whether the *grantee* only read access to the data of the *Access Grantors* has or whether the *grantee* the complete account of the *Access Grantors* may take over.
- Waiting time: Defines how long the *grantee* has to wait until he accesses the data of the *Access Grantor* can access after the *Access Grantor* a *Emergency Access* This waiting period gives the *Access Grantor* a time window in which the *Access Grantor* the *Emergency Access* can revoke.

More information about Vaultwarden *Emergency Access* can be viewed at the following link:

https://bitwarden.com/help/emergency-access/

As soon as the *Access Grantor* a *Emergency Access* created, the metadata of the *Emergency Access* via the API endpoint /api/emergency-access/<access_UUID> This allows the *Access Grantor*, the conditions for the created *Emergency Access* to change. Including the *Access Level* and the waiting time.

Vaultwarden has not implemented any access control on this API endpoint for modifying the metadata. This means that even an unauthenticated user could modify the metadata of a *Emergency Access* change if the UUID of the

*Emergency Access*is known. The following screenshot illustrates the problem.

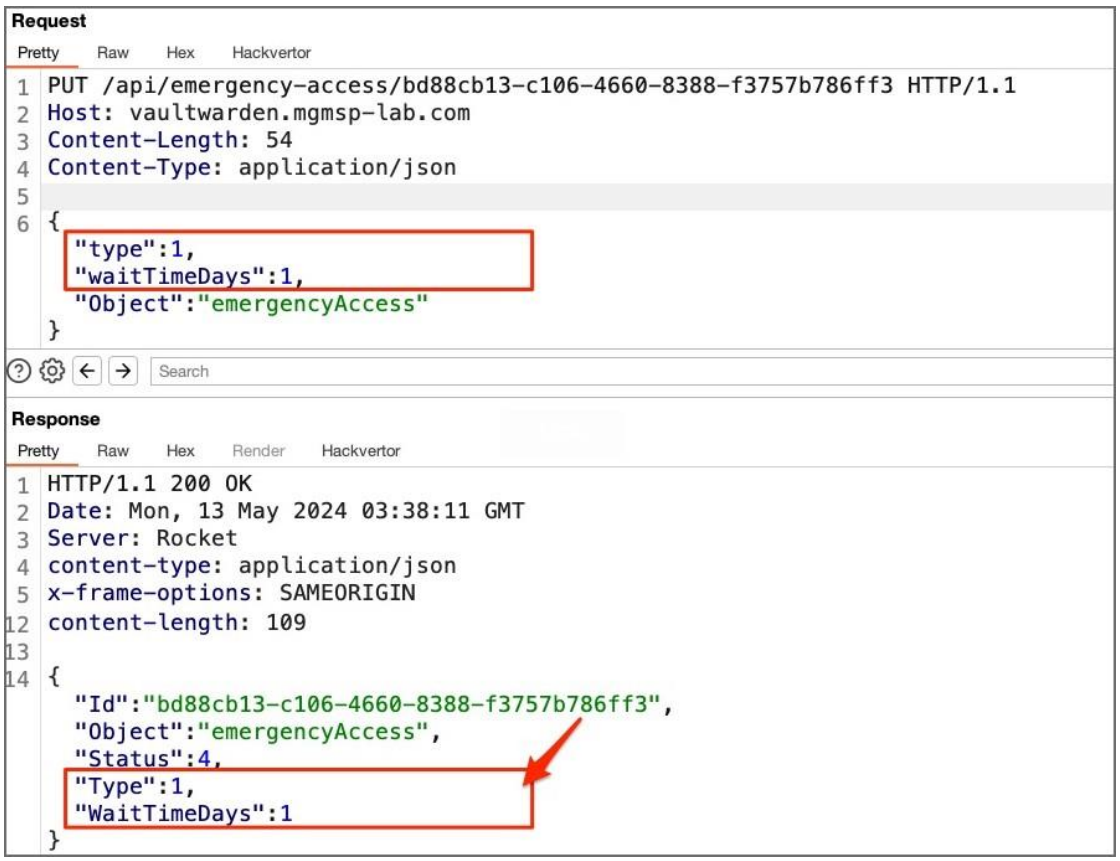


Figure 8: Missing authentication check in the Emergency Access metadata endpoint

To view the metadata of a*Emergency Access*To change the UUID of the *Emergency Access*Vaultwarden uses UUID version 4 for entity IDs that are difficult to enumerate. For an attack by the*grantee*However, this is irrelevant as he will receive this information in the corresponding notification email.

Figure 9: Notification email when designated as a grantee

exploitability

As soon as a *Emergency Access* is set up, the *grantee* the Change access conditions, including the *Access Level* and the waiting time. This allows the *grantee* on the data of the *Access Grantor* with a higher access level. This means that the account can be completely taken over instead of read-only access without having to wait for the specified waiting period (7 days by default). This will ensure that the *Access Grantor* specified access restrictions are ineffective.

Note: The vulnerability only affects the metadata endpoint of the *Emergency Access*. The endpoint for creating a *Emergency Access* checks the required rights correctly. The exploitability of this vulnerability is therefore limited to the *grantee* higher privileges for accessing the data of the *Access Grantor* can give.

vulnerability in the code

- Source: https://github.com/danigarcia/vaultwarden/blob/1.30.3/src/api/core/emergency_access.rs#L115-L148

1.30.3 vaultwarden / src / api / core / emergency_access.rs ↑ Top

Code Blame Raw

```rust
#[put("/emergency-access/<emer_id>", data = "<data>")]
async fn put_emergency_access(emer_id: &str, data: JsonUpcase<EmergencyAccessUpdateData>, conn:
    post_emergency_access(emer_id, data, conn).await
}

#[post("/emergency-access/<emer_id>", data = "<data>")]
async fn post_emergency_access(
    emer_id: &str,
    data: JsonUpcase<EmergencyAccessUpdateData>,
    mut conn: DbConn,
) -> JsonResult {
    check_emergency_access_enabled()?;

    let data: EmergencyAccessUpdateData = data.into_inner().data;

    let mut emergency_access = match EmergencyAccess::find_by_uuid(emer_id, &mut conn).await {
        Some(emergency_access) => emergency_access,
        None => err!("Emergency access not valid."),
    };

    let new_type = match EmergencyAccessType::from_str(&data.Type.into_string()) {
        Some(new_type) => new_type as i32,
        None => err!("Invalid emergency access type."),
    };

    emergency_access.atype = new_type;
    emergency_access.wait_time_days = data.WaitTimeDays;
    if data.KeyEncrypted.is_some() {
        emergency_access.key_encrypted = data.KeyEncrypted;
    }

    emergency_access.save(&mut conn).await?;
    Ok(Json(emergency_access.to_json()))
}
```

Figure 10: Vulnerability in the code

measure

Vaultwarden must correctly check the required access rights at the metadata endpoint, as already done at other API endpoints. Only the *Access Grantor* should the metadata of a *Emergency Access* may change.

E2 access control

threat

While authentication verifies the identity of a user (or technical entity), access control enforces policies that prevent users from acting outside of their intended permissions. Missing or incorrectly implemented

Authorization checks can facilitate unauthorized access to resources, often resulting in information disclosure or unwanted modification of data.

A lack of authorization also often results in the escalation of privileges, which generally means unauthorized access to higher privileges within a system.

E2.1 Lack of rotation of organizational keys [high]

observation

Vaultwarden allows members of an organization to share data (*Secrets*) with each other. To do this, all data of an organization is encrypted with a symmetric key (the organization key) before it is stored on the server. Each member of an organization receives the
Organization key during the onboarding process via a key sharing mechanism.

However, Vaultwarden does not have an offboarding process for members who leave an organization. This means that the organization key is not rotated when a member leaves an organization. As a result, the departing member, who was essentially deprived of access, still owns the organization key that was used.

The lack of key rotation can be recreated with the following steps:

1. Create an organization with multiple members and *Secrets*
2. Note the encrypted *Secrets* that are stored on the server
3. Revoking a member’s access to the organization
4. The encrypted data noted in step 2 *Secrets* were not changed after step 3

By exploiting the E2.2 vulnerability, a user who has left an organization can still access the encrypted *Secrets* of the organization. However, since the user is still in possession of the organization key due to the lack of key rotation, he can decrypt the encrypted data retrieved via E2.2. This gives him unauthorized access to the *Secrets* of the organization. This unauthorized access also includes *Secrets* that were created or updated after the user left the organization.

The attack is demonstrated as an example below.

First, a test user (*Mallory*-the attacker) access to an organization,*Foobar*, granted.

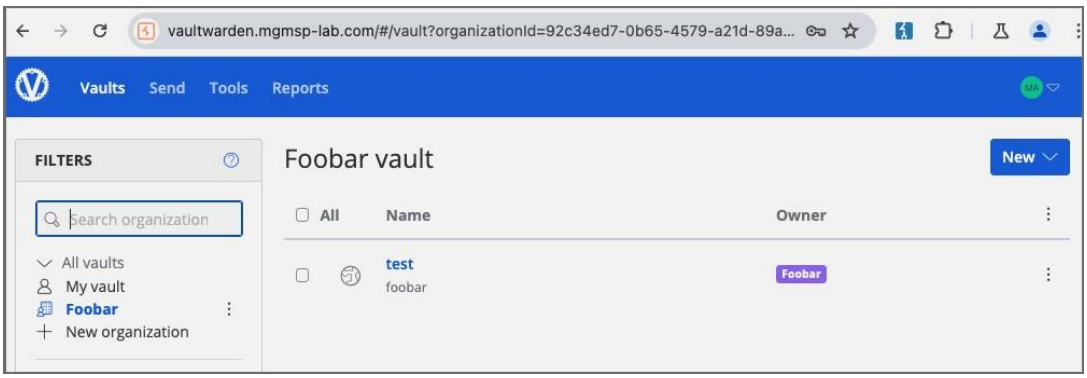


Figure 11: Mallory gains access to the organization Foobar

Through the onboarding process,*Mallory*the organization key. She notes the organization ID and the (encrypted) organization key from the response of the GET endpoint `/api/sync`.

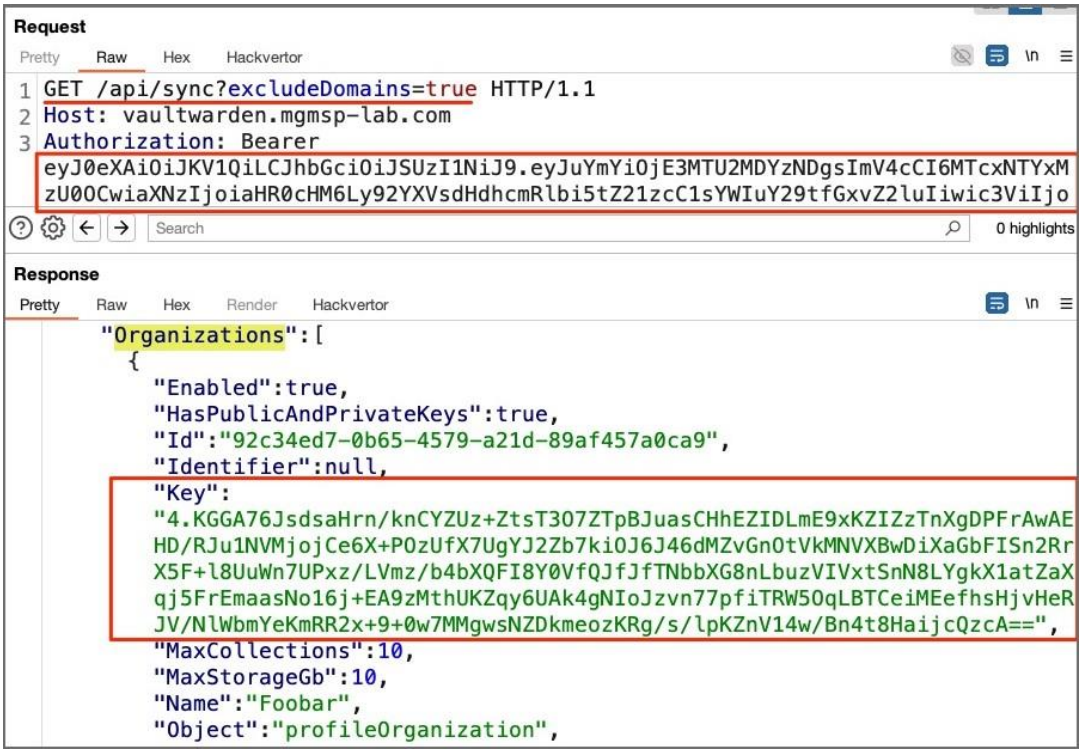


Figure 12: Mallory notes the organization ID and organization key

Next removed*Alice*(an organizational administrator)*Mallory*from the *Foobar* organization. This has*Mallory*no longer have access to the*Secrets*the organization.

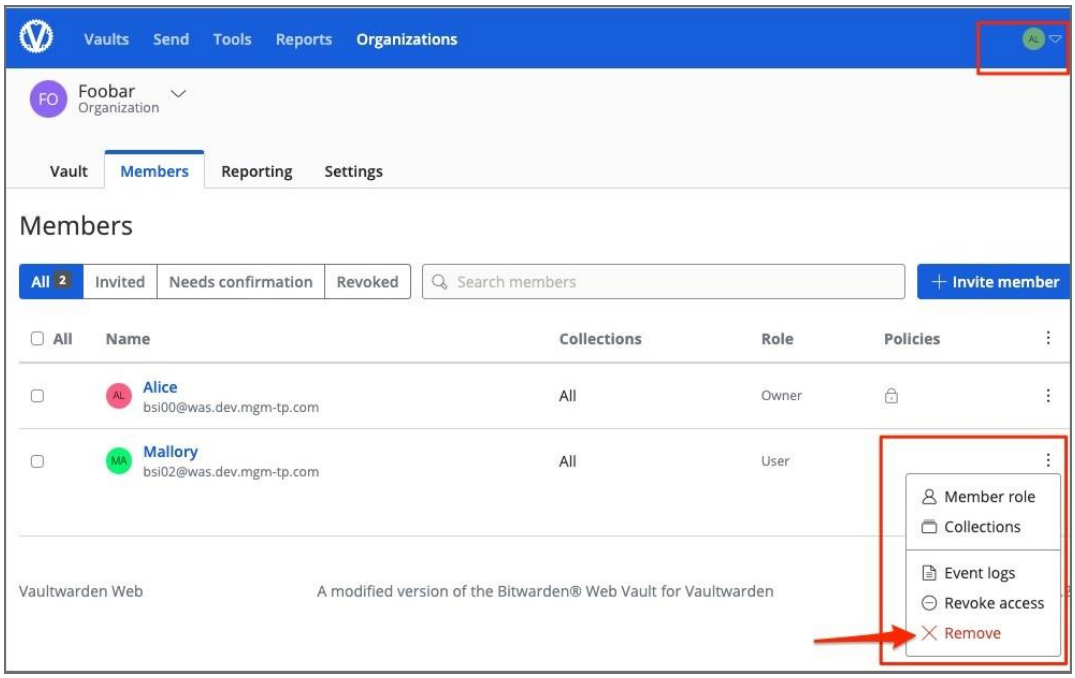


Figure 13: Organization admin removes Mallory's access

Next, add *Alice* the *Foobar* organization adds new secrets.

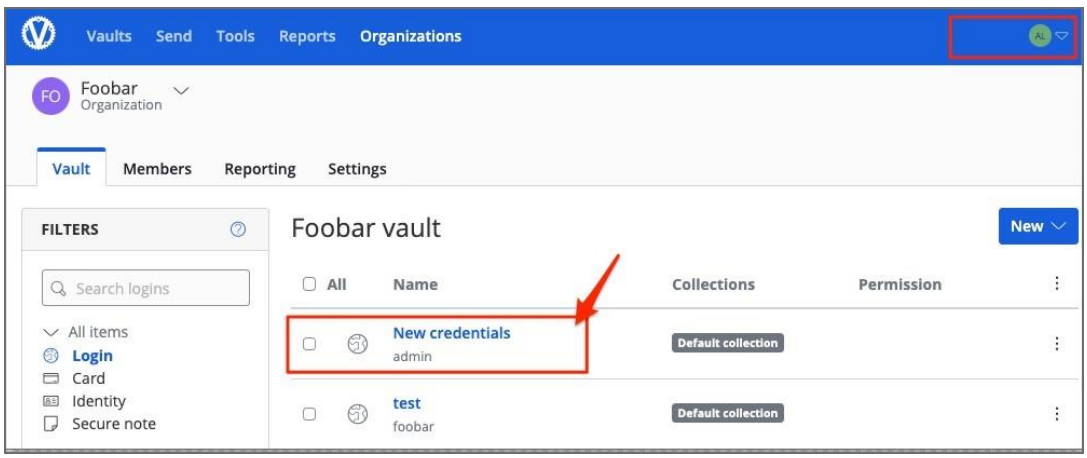


Figure 14: A new secret is added to the Foobar organization vault

It can be confirmed via the Vaultwarden user interface that *Mallory* no longer have access to the *Secrets* the *Foobar* organization should have.

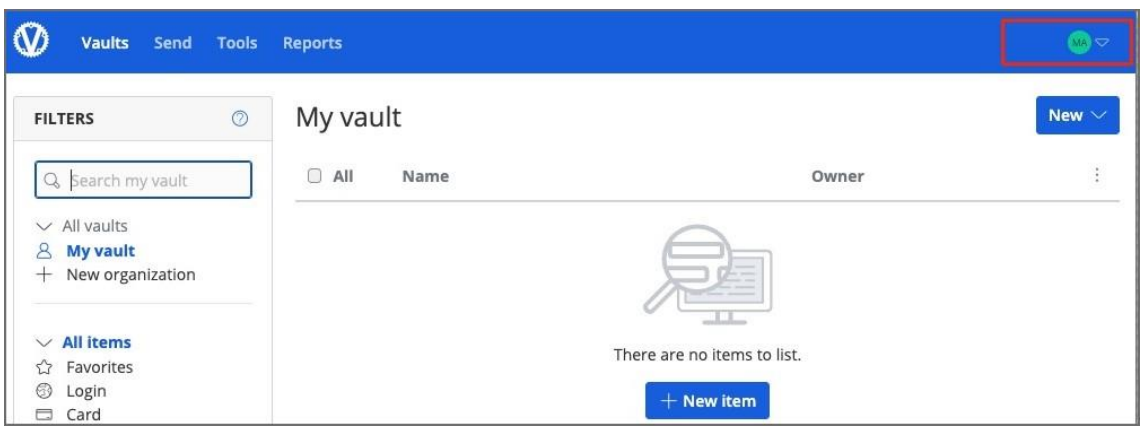


Figure 15: Mallory lacks access to the Foobar organization

Mallory now exploits the vulnerability E2.2 and thus gains possession of the newly added (encrypted) *Secrets* the *Foobar*-Organization.

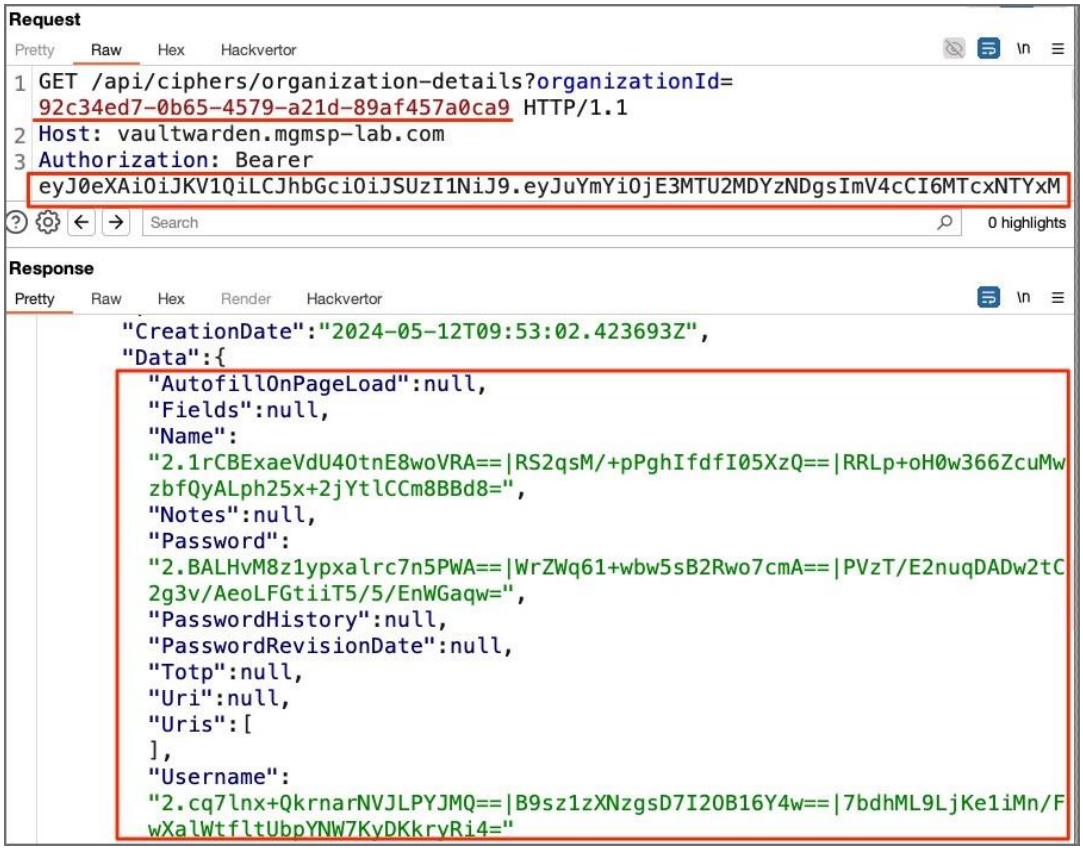


Figure 16: Mallory exploits “Unauthorized access to encrypted corporate data”

Mallory can use the encrypted data obtained in this way *Secrets* decrypt it again using the noted organizational key. This can be done in various ways:

1. Based on the Bitwarden whitepaper, *Mallory* decrypt the encrypted organization key with their personal key and then use the plaintext organization key to decrypt the organization data.

2nd *Mallory* could manipulate their own traffic with the application and thus make the application decrypt the data for them. (Note: Since the decryption is then performed client-side, the decryption could also be performed directly via the local JavaScript.)

For a proof of concept, method #2 is used.

Mallory refreshes the application and manipulates the response to the synchronization requests `/api/sync`. In the following screenshot the parameter `Cipher` of the original request is empty because Mallory does not have access to a secret (see above). Since she also has access to the *Foobar* organization, the field is also `Organizations` empty. *Mallory* These two parameters must be manually fill the intercepted request with data from the previous steps.

- `Ciphers` : Contains the array of encrypted data that *Mallory* by exploiting the E2.2 vulnerability.
- `Organizations` : Contains the metadata of the *Foobar* organization that *Mallory* from the time when she was still a member of the organization.

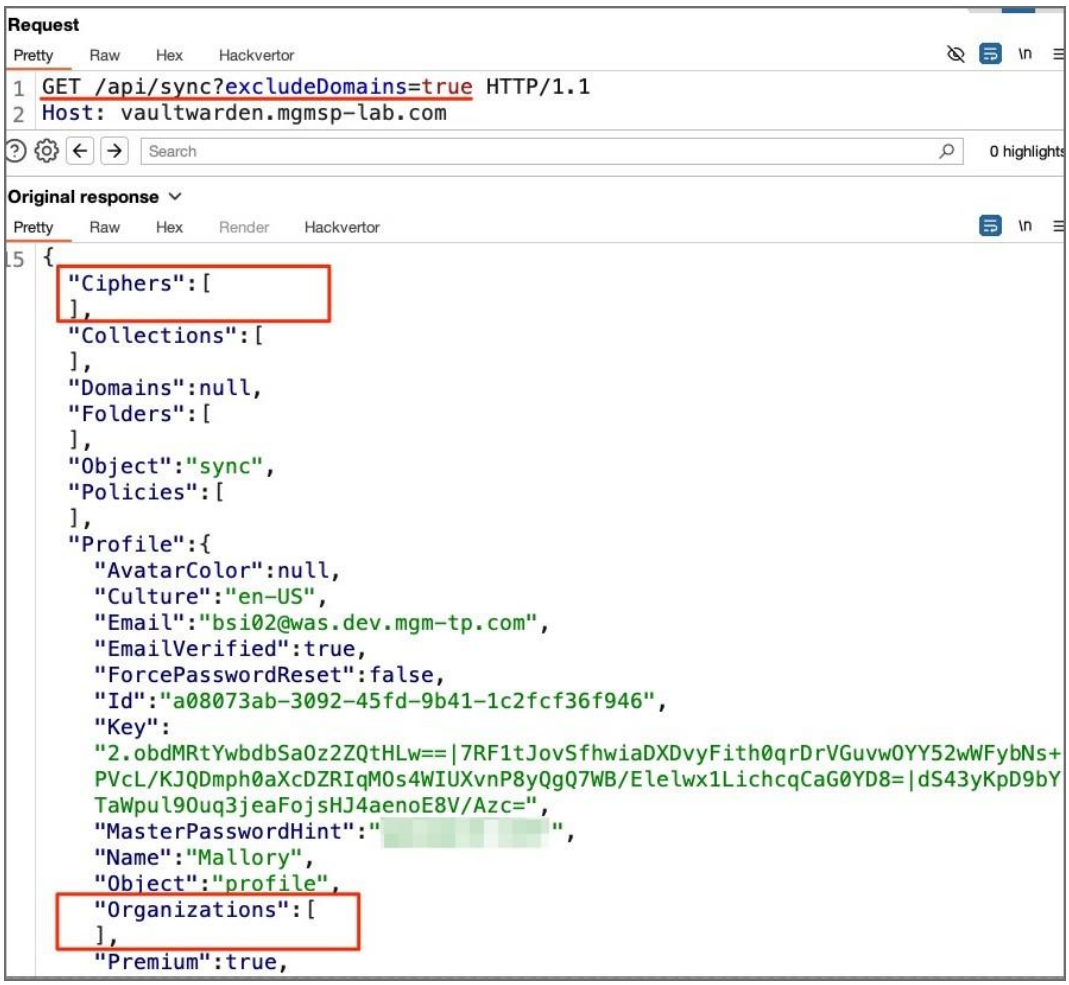


Figure 17: Parameters that must be manually filled in the intercepted request

After the required parameters have been replaced accordingly, the response is forwarded to a client. The client then automatically decrypts the data. Finally, *Mallory* the decrypted data in the user interface, with all *Secrets* the *Foobar*-Organization.

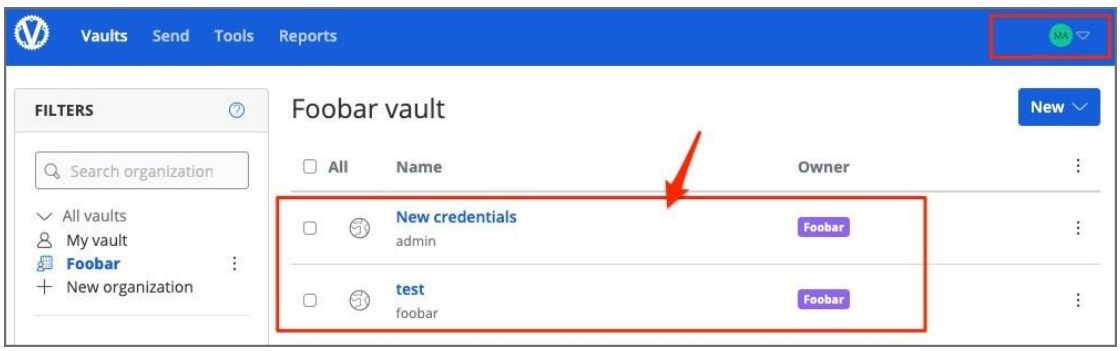


Figure 18: Mallory sees all of Foobar’s secrets in plain text

exploitability

Users who have been denied access to an organization can still access the *Secrets* this organization. Including access to *Secrets* that were added after the time of withdrawal of access.

vulnerability in the code

The lack of rotation of the keys cannot be proven with a concrete code snippet. Therefore, despite the criticality *[high]* No source is given in the code for this finding.

measure

If an organization administrator changes the access rights of a member, i.e. revokes access to a member, the organization data must be re-encrypted with a newly generated key (key rotation of the organization key). This process must be performed by the client of the
organization administrator making the change. The new organization key must be synchronized with the other organization members through the existing key exchange mechanism.

E2.2 Unauthorized access to encrypted data [medium]

observation

Vaultwarden does not adequately protect some encrypted data stored on the server.

An authenticated user could thus gain unauthorized access to encrypted data of any organization, even if the user is not a member of the targeted organization. However, the user would have to be provided with the appropriate `organizationId` This is shown in the following screenshot of the recognize. The user "*flapper*" bearer token should not actually have access to the information shown.

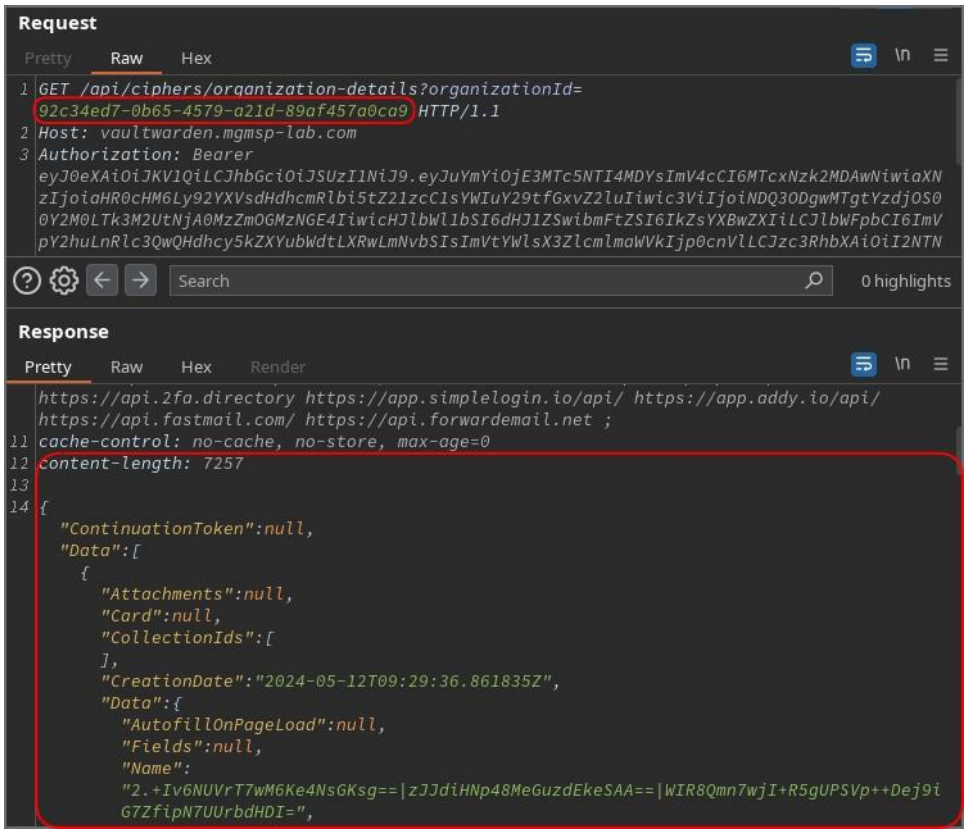


Figure 19: Return of organizational data without appropriate authorization

In addition, the API endpoint `/api/organizations/< organizationId > /keys` , which is next to the*public key*also the encrypted*private key*the organization, returns the requested data without prior authentication of the user.

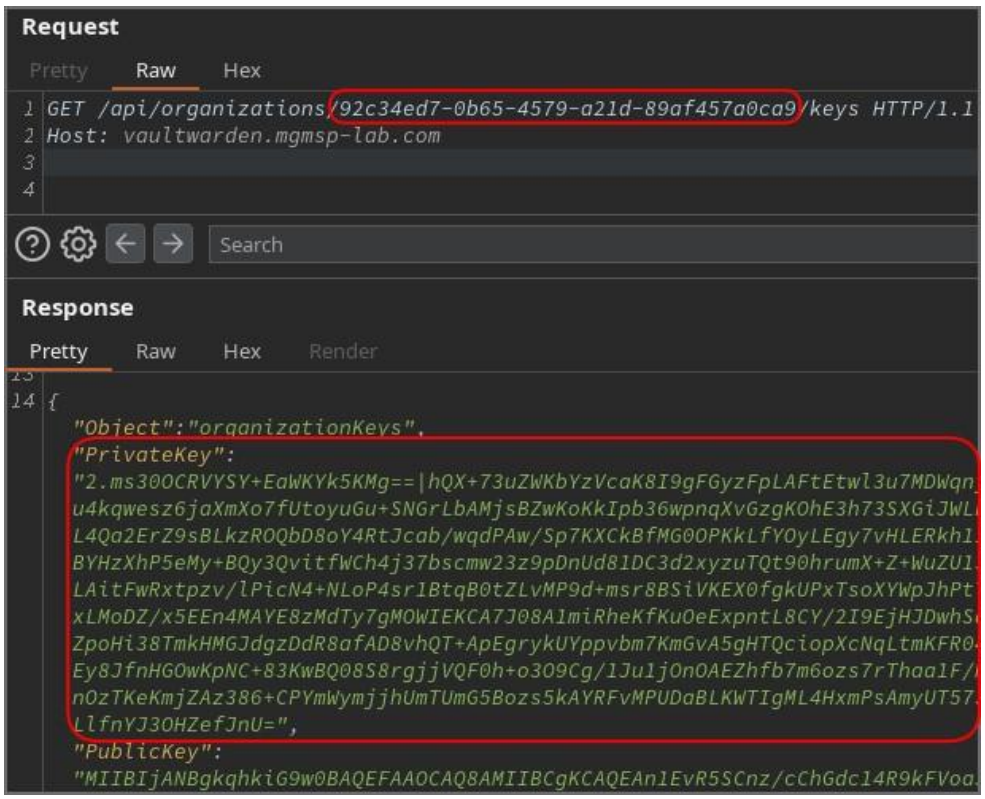


Figure 20: Returning the encrypted private key without authentication

Note: The requested, unauthorized data is encrypted and therefore not actually readable by an attacker (see Finding E2.1)

exploitability

The leaked data is encrypted using strong encryption (AES 256-bit encryption with cryptographically secure, randomly generated keys). This means that it is practically impossible to decrypt it using only the exposed data. Not even in an offline brute force attack.

However, Vaultwarden does not rotate the organizational keys used for encryption (see Finding E2.1). In the specific case where a user has been deprived of access to the company data but already has the organizational key, they could use the keys to decrypt the leaked data.

Note: Due to the increased exploitability of the vulnerability through the finding E2.1, as well as the special protection needs of Vaultwarden with regard to the management of secrets, this finding is assigned the criticality *medium* classified.

measure

Access control must be set at the API endpoint `/api/ciphers/organization-details` as implemented in the rest of the application to ensure that only data is returned for which the requesting user has the appropriate access authorization.

The encrypted *private key* should be derived from the API endpoint response `/api/organizations/< organizationId > /keys` be removed. The *Private Key* should only be returned via another API endpoint with appropriate access control.

E2.3 Possible hardening measures of the Docker environment [low]

observation

When using the official Vaultwarden Docker image, the started server service is started with the user `root` executed.

https://hub.docker.com/r/vaultwarden/server

```
[redacted]@BSI-CAOS3:~$ sudo docker container top vaultwarden -o user,pid,command,%cpu,%mem
USER                PID                 COMMAND             %CPU                %MEM
root                379887              /vaultwarden        0.0                 1.6
```

Figure 21: Vaultwarden process executed by root user

This behavior can also be confirmed by analyzing the Vaultwarden Docker files:

https://github.com/dani-garcia/vaultwarden/blob/main/docker/Dockerfile.debian

exploitability

If no USER is specified in the Dockerfile, processes within the container are started withrootrights. The execution of applications within

the Docker container withrootRights increases the attack surface and makes it easier to exploit if the system is successfully compromised.

measure

Docker containers should not be under therootuser, but a user with restricted rights.

The Vaultwarden server service can be run without root privileges. The relevant documentation can be found at the following link.

https://docs.docker.com/develop/develop-images/instructions/#user

E3 cross-site scripting

threat

Cross-site scripting (XSS) generally refers to the exploitation of a vulnerability by injecting data from one context in which it is not trusted, such as user input, into another context in which it is trusted. For example, malicious JavaScript code can be embedded in HTML pages without being checked and executed in the context of a user's browser.

XSS vulnerabilities can be exploited in many ways, for example:

- Session hijacking (stealing the session ID and thus breaking into another user's account by bypassing the login).

- Website spoofing (fake website) for phishing (stealing login data and personal information) or other forms of deception and fraud. The original URL of the website is still displayed and the certificate in an HTTPS connection still attests to the authenticity of the site - it remains unharmed.

-Spying on or manipulating data on the user’s system.

- Attack on the browser to gain complete control of the client system using freely available attack tools (e.g. BeEF).

There are different forms of cross-site scripting.**Stored Cross-Site Scripting**The malicious code is stored by the attacker within the application and is executed every time a user accesses the stored data.

At the**Reflected cross-site scripting**The malicious code is not saved, but embedded directly in the server response. The easiest way to carry out such an attack is to send an email with a link that the user must click on in order for the attack to be carried out. With certain email client settings, this click is not even necessary in some cases.

A special form is**DOM-based cross-site scripting**, in which the malicious code is not embedded and executed by the server but on the client side by modifying the DOM.

Due to the current prevalence of phishing attacks and the possibility of session hijacking, we generally rate XSS vulnerabilities as [high]. However, if exploitation is only possible under certain circumstances, the risk is reduced. However, remediation is still strongly recommended, as it cannot be ruled out that there are still opportunities for easy exploitation.

E3.1 HTML injection possible [medium]

observation

Vaultwarden comes with an admin dashboard. This functionality can be enabled following the instructions described here:

https://github.com/dani-garcia/vaultwarden/wiki/Enabling-admin-page

Once activated, the Vaultwarden Admin Dashboard is available at the path. The admin dashboard contains a user management page. This allows an administrator to, for example, change the role assignment for users who are members of an organization via a dialog.

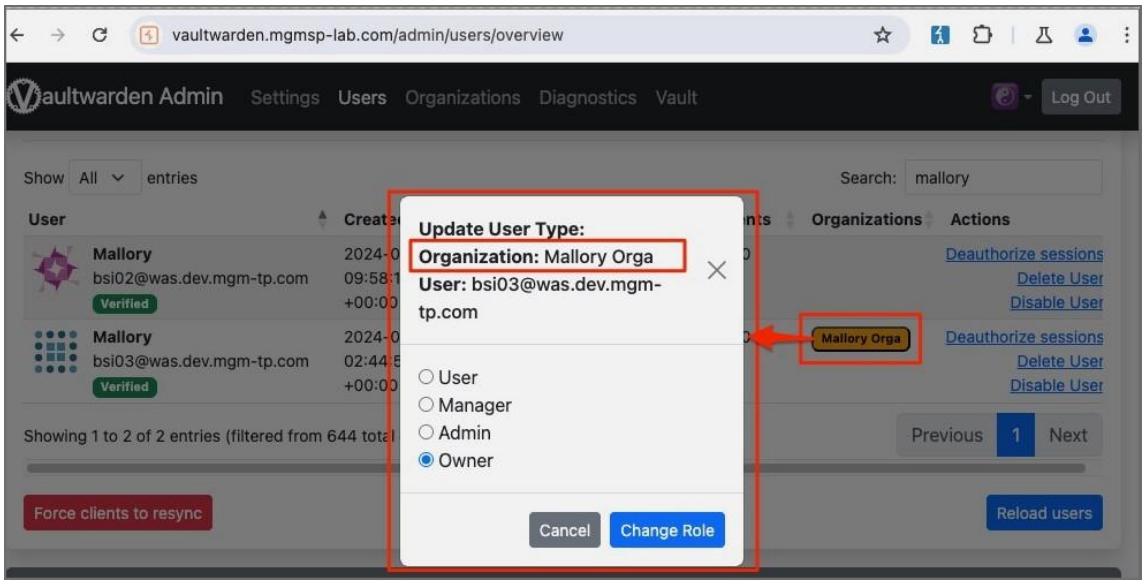


Figure 22: User management - Role assignment dialog

This dialog is vulnerable to HTML injection. Payloads can be injected into the organization name and email address fields. As a proof of concept, the following payload was stored in the organization name. Since there are length restrictions in the frontend, the request must be intercepted, e.g. via the Burp proxy, and modified accordingly.

```
<div style=position:absolute;width:100%;height:100%;background:red;zindex:9>Foobar</div>
```

The payload in the organization name is then parsed as HTML code in the user management dialog and rendered accordingly.

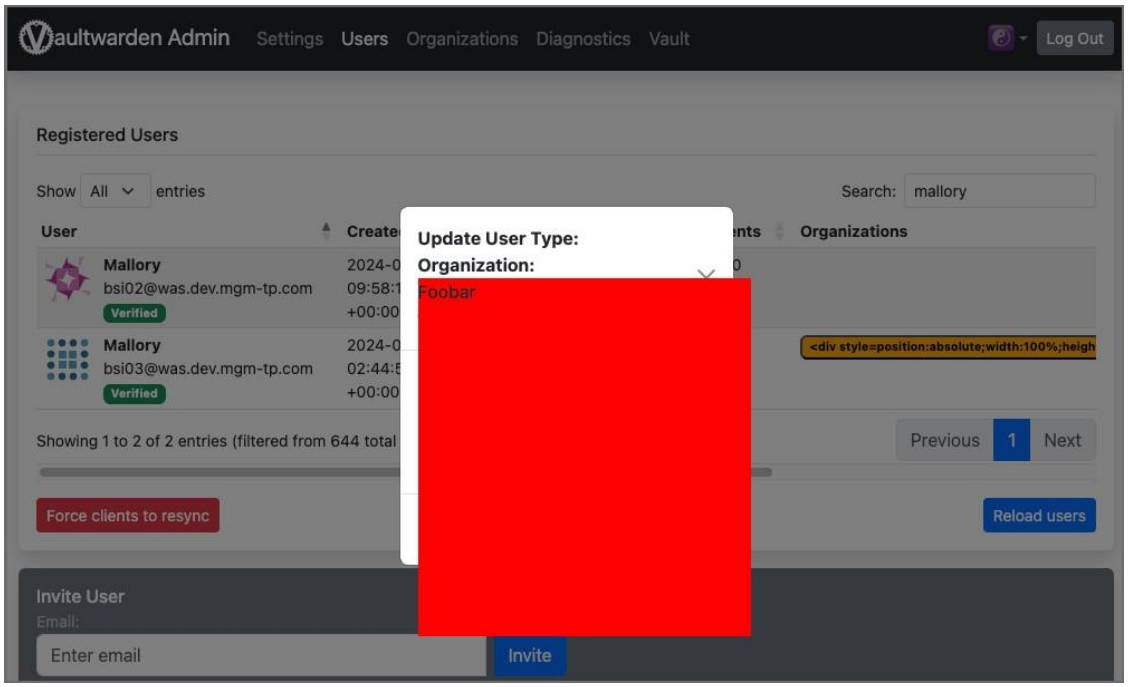


Figure 23: Injected HTML code

By default, Vaultwarden uses a strict *content security policy* (CSP). This CSP successfully prevented attempts to execute JavaScript via the vulnerability. An escalation of the exploitation of this vulnerability from HTML injection to cross-site scripting (XSS) was therefore not possible during the penetration test. However, the following payload can be used to redirect affected users to arbitrary other pages when the dialog is called.

```
<meta name=language content=1;https://mgm-sp.com HTTP-EQUIV=refresh />
```

The application's default CSP is defined as follows.

```
content-security-policy: default-
src 'self';
base-uri 'self';
form-action 'self';
object-src 'self' blob:; script-src 'self' 'wasm-
unsafe-eval'; style-src 'self' 'unsafe-inline';

child-src 'self' https://*.duosecurity.com https://*.duofederal.com; frame-src 'self' https://
*.duosecurity.com https://*.duofederal.com; frame-ancestors 'self' chrome-

extension://nngceckbapebfimnlniiiahkandclblb chromeextension://
jbkfoedolllekgbhcbcoahefnbanhhlh moz-extension://* ; img-src 'self' data: https://
haveibeenpwned.com ;
connect-src 'self' https://api.pwnedpasswords.com https://api.2fa.directory
https://app.simplelogin.io/api/ https://app.addy.io/api/ https: //
api.fastmail.com/ https://api.forwardemail.net ;
```

Note: The finding was identified by the Semgrep Pro SAST analysis (see D1.2) and confirmed by the dynamic analysis.

exploitability

HTML injection vulnerabilities are a sign of a lack of input validation. By inserting HTML tags, it is possible to change the appearance and content of the page and embed links to malicious pages, or possibly execute scripts.

Since the Admin Dashboard authenticates requests using cookies (see Finding E8.1), an embedded HTML form with additional user interaction could also be used to call other state-changing API endpoints available in the Admin Dashboard.

Note: The exploitability of this finding is limited to HTML injection. However, since a *Open-Redirect,* which could affect administrators of the application, the finding is rated with criticality [medium].

vulnerability in the code

- Source: https://github.com/danigarcia/vaultwarden/blob/1.30.3/src/static/scripts/admin_users.js#L201

src > static > scripts > JS admin_users.js > updateUserOrgType

```
189   const userOrgTypeDialog = document.getElementById("userOrgTypeDialog");
190   // Fill the form and title
191   userOrgTypeDialog.addEventListener("show.bs.modal", function(event) {
192       // Get shared values
193       const userEmail = event.relatedTarget.parentNode.dataset.vwUserEmail;
194       const userUuid = event.relatedTarget.parentNode.dataset.vwUserUuid;
195       // Get org specific values
196       const userOrgType = event.relatedTarget.dataset.vwOrgType;
197       const userOrgTypeName = ORG_TYPES[userOrgType]["name"];
198       const orgName = event.relatedTarget.dataset.vwOrgName;
199       const orgUuid = event.relatedTarget.dataset.vwOrgUuid;
200
201       document.getElementById("userOrgTypeDialogTitle").innerHTML = `<b>Update User Type:</
          b><br><b>Organization:</b> ${orgName}<br><b>User:</b> ${userEmail}`;
202       document.getElementById("userOrgTypeUserUuid").value = userUuid;
203       document.getElementById("userOrgTypeOrgUuid").value = orgUuid;
204       document.getElementById(`userOrgType${userOrgTypeName}`).checked = true;
205   }, false);
206
```

Figure 24: Source code of the function responsible for HTML injection.

measure

User input must be validated on the server side before input and encoded in the respective context before output. In this case, untrusted user input (organization name, user email) must be HTML encoded before being embedded in the final HTML code.

E4 Server Side Request Forgery (SSRF)

Server-side request forgery can occur whenever a web application itself acts as a client, i.e. it calls another application or service and uses parameters that it receives from the external client (browser, app, fat client, etc.) and does not sufficiently validate them.

In this case, the external client can manipulate the communication between the server and the called service or even access other services. These can be external services (on the Internet), internal services (on the intranet) or local services (on the server itself).

Example:

An application is called as follows: http://example.com/action.do?update=backend/check. The application uses the value of the update parameter to generate a service call, e.g. according to this pattern: http://backend/check.

Since an attacker can assign any value to the parameter, validation must be carried out at the point of use. If this is not the case, the string exchanged by the attacker would, for example, http://example.com/action.do?update=10.1.2.3:8123/get_account_balance the server-side call http://10.1.2.3:8123/get_account_balance must be made.

Notice

In many cases, SSRF vulnerabilities cannot be detected using penetration testing. This means that the fact that no vulnerability has been found does not mean that this vulnerability does not exist. In order to make a reliable negative statement, an analysis of the relevant code sections is usually necessary.

E4.1 Server-Side Request Forgery possible [low]

observation

In the Vaultwarden server, two *server-side request forgery* vulnerabilities are identified.

In the default installation, Vaultwarden uses an internal *icon service* to download favicons from external websites to the Vaultwarden server. These icons will then be displayed in the Vaultwarden user interface. An icon download can be initiated by making an unauthenticated request to the API endpoint `/icons/<domain>/icon.png` be triggered.

By setting the path parameter `<domain>` to an attacker-controlled domain, ...

```
GET /icons/suqgizc607wzjk9ke3w2kpwp6gc70xom.mgmsp-lab.com/icon.png HTTP/1.1 Host: vaultwarden.mgmsp-lab.com
```

... a request is sent from the server to the target specified by the attacker.

#	Time	Type	Payload	Source IP address
1	2024-Apr-16 03:19:07.792 UTC	HTTP	suqgizc607wzjk9ke3w2kpwp6gc70xom	185.40.248.10
2	2024-Apr-16 03:19:07.607 UTC	DNS	suqgizc607wzjk9ke3w2kpwp6gc70xom	172.253.193.194
3	2024-Apr-16 03:19:07.608 UTC	DNS	suqgizc607wzjk9ke3w2kpwp6gc70xom	172.217.44.133
4	2024-Apr-16 03:19:07.871 UTC	HTTP	suqgizc607wzjk9ke3w2kpwp6gc70xom	185.40.248.10

Description | Request to Collaborator | Response from Collaborator

Pretty | Raw | Hex | Hackvertor

```
1  GET /favicon.ico HTTP/1.1
2  referer: https://suqgizc607wzjk9ke3w2kpwp6gc70xom.mgmsp-lab.com/
3  accept: text/html, text/*;q=0.5, image/*, */*;q=0.1
4  user-agent: Links (2.22; Linux X86_64; GNU C; text)
5  accept-language: en,*;q=0.1
6  cache-control: no-cache
7  pragma: no-cache
8  accept-encoding: gzip, br
9  host: suqgizc607wzjk9ke3w2kpwp6gc70xom.mgmsp-lab.com
10
11
```

Vaultwarden checks the passed parameter `<domain>` and rejects the Request if the domain resolves to an internal IP. Validation can be confirmed via the Vaultwarden log file. In the following example, the request is sent to the host `localhost` correctly rejected.

```
[redacted]@BSI-CAOS3:/var/www/vw-data$ tail -n 10 vaultwarden.log
[2024-05-14 04:55:42.849][request][INFO] GET /alive
[2024-05-14 04:55:42.851][response][INFO] (alive) GET /alive => 200 OK
[2024-05-14 04:56:20.306][request][INFO] GET /icons/localhost/icon.png
[2024-05-14 04:56:20.309][vaultwarden::api::icons][DEBUG] IP 127.0.0.1 for domain 'localhost'
 is not a global IP!
[2024-05-14 04:56:20.309][vaultwarden::api::icons][WARN] Unable to download icon: Host resolv
es to a non-global IP. localhost
[2024-05-14 04:56:20.310][response][INFO] (icon_internal) GET /icons/<domain>/icon.png => 200
 OK
```

Figure 25: Rejected request to localhost

However, the validation can be bypassed if an approved (external) domain is passed that redirects to a blocked internal domain. It is also possible to redirect directly to an internal IP. Vaultwarden then follows the redirect. In the following proof of concept, when calling from the external domain `alcatraz.mgm-sp.team/favicon.icon` on `http://localhost/admin` forwarded.

```
→  CAOS3 curl https://alcatraz.mgm-sp.team/favicon.ico -i
HTTP/1.1 302 Found
Date: Tue, 14 May 2024 06:44:08 GMT
Server: Apache
Location: http://localhost/admin
Content-Length: 206
Content-Type: text/html; charset=iso-8859-1

<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<html><head>
<title>302 Found</title>
</head><body>
<h1>Found</h1>
<p>The document has moved <a href="http://localhost/admin">here</a>.</p>
</body></html>
```

Figure 26: Redirect to an internal domain

If the favicon download is triggered from the prepared external domain, the Vaultwarden server followed the redirect and then sent a request to the specified internal host. The behavior can be confirmed via the Vaultwarden log file:

```
[redacted]@BSI-CAOS3:/var/www/vw-data$ tail -n 22 vaultwarden.log

[2024-05-14 06:55:04.241][reqwest::async_impl::client][DEBUG] redirecting 'https://alcatraz.mgm-sp
.team/favicon.ico' to 'http://localhost/admin'
[2024-05-14 06:55:04.243][reqwest::connect][DEBUG] starting new connection: http://localhost/
[2024-05-14 06:55:04.244][request][INFO] GET /admin
[2024-05-14 06:55:04.244][vaultwarden::api::admin::_][WARN] Request guard `AdminToken` is forwardi
ng.
[2024-05-14 06:55:04.245][response][INFO] (admin_page_login) GET /admin/ [2] => 200 OK
[2024-05-14 06:55:04.246][vaultwarden::api::icons][DEBUG] Icon from https://alcatraz.mgm-sp.team/f
avicon.ico, is not a valid image type
```

Figure 27: Following the redirect to an internal host

Vaultwarden checks the response of the server-side request. Only if the response points to image content (starting with the file signature or the *Magic Bytes* an image), the content is returned to the user.

In addition, API endpoint for setting up Duo 2FA authentication (`/api/two-factor/duo`) susceptible to *server-side request forgery*. The target host can, as in Screenshot to see the parameter `host` be specified.

Request

Pretty Raw Hex Hackvertor

```
1 PUT /api/two-factor/duo HTTP/1.1
2 Host: vaultwarden.mgmsp-lab.com
3 Authorization: Bearer
  eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9.eyJuYmYiOjE3MTU2NzU4MTQsImV4cCI6MTcxNTY4M
4 Connection: close
5 Content-Length: 183
6
7 {
    "masterPasswordHash":"[redacted]",
    "integrationKey":"integrationkey",
    "secretKey":"secretkey",
    "host":"suqgizc607wzjk9ke3w2kpwp6gc70xom.mgmsp-lab.com"
  }
```

Figure 28: SSRF over Duo 2FA authentication

The above request then leads to the following request sent by the Vaultwarden server.

#	Time	Type	Payload	Source IP address
19	2024-May-14 08:58:27.119 UTC	HTTP	suqgizc607wzjk9ke3w2kpwp6gc70xom	185.40.248.10
18	2024-May-14 08:37:09.880 UTC	DNS	suqgizc607wzjk9ke3w2kpwp6gc70xom	172.253.225.37

Description Request to Collaborator Response from Collaborator

Pretty Raw Hex Hackvertor

```
1 GET /auth/v2/check HTTP/1.1
2 authorization: Basic aW50ZWdyYXRpb25rZXk6MDQwNjYxZDI4MTI5NTU1N2FkZWRlZDE3ZmRiN
3 user-agent: vaultwarden:Duo/1.0 (Rust)
4 date: Tue, 14 May 2024 08:58:27 +0000
5 accept: */*
6 accept-encoding: gzip, br
7 host: suqgizc607wzjk9ke3w2kpwp6gc70xom.mgmsp-lab.com
8
```

Figure 29: Received SSRF request

exploitability

server-side request forgery(SSRF) allows an attacker to trigger certain HTTP requests through the server. Further exploitability depends heavily on the other internal services running in the internal network or on the server of the application being tested.

An attacker could use this vulnerability to issue HTTP GET requests to any internal endpoint. However, the application will only pass the content of the SSRF attack response to the user if the response content is an image.

Note: Due to this limitation, this is an (almost) blind SSRF vulnerability. Therefore, this finding is only rated with criticality [low].

measure

In case of a redirect response from an external server, Vaultwarden should double-check the new target host before redirecting to make sure it is not part of the internal network.

Alternatively, Vaultwarden could reject redirects altogether if the redirect leads to a new host.

E5 Session ID in the URL or referrer (referrer leak)

If the session ID is transported via a cookie, it should never be transported via the URL. This creates additional opportunities for spying on it:

- In the user's browser (history, possibly bookmarks, "looking over the shoulder") for those who have physical access to the user's browser
- Copying and sending page excerpts containing the link with the session ID
- Server logs/proxy logs that may be accessible to third parties Via the
- referrer header, which contains the URL of the page on which the click occurred or from which images were loaded, etc. If these are external websites, the session ID ends up in the hands of third parties via the referrer.

E5.1 Transmission of sensitive data in the URL [low]

observation

After a successful login, the web application also establishes a websocket connection to the Vaultwarden server. The corresponding request is authenticated with the user's JWT transported in the URL.

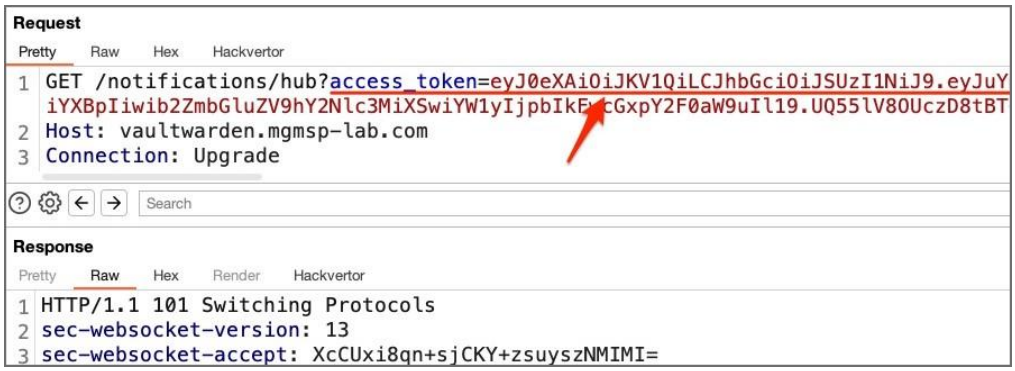


Figure 30: Transport of the JWT in the URL when changing protocol

Consequently, the user's full JWT is logged, for example, in the log of the Apache server used.

```
[redacted]@BSI-CAOS3:~$ sudo cat /var/log/apache2/access.log | grep access_token | tail
10.54.20.25 - - [14/May/2024:04:11:17 +0000] "GET /notifications/hub?access_token=eyJ0eXAiOiJKV1QiLC
.eyJuYmYiOjE3MTU2NTU5NDQsImV4cCI6MTcxNTY2MzE0NCwiaXNzIjoiaHR0cHM6Ly92YXVsdHdhcmRlbi5tZ21zcC1sYWIuY2
joiYTA4MDczYWItMzA5Mi00NWZkLTliNDEtMWMyZmNmMzZmOTQ2IiwicHJlbWl1bSI6dHJ1ZSwibmFtZSI6Ik1hbGxvcnkiLCJlb
hcy5kZXYubWdtLXRwLmNvbSIsImVtYWlsX3Zlcmlmaed2ZkIjp0cnVlLCJzc3RhbXAiOiI4ODgwM2Y0NC1jNzBiLTQzZDctYjI1YS1
CJkZXZpY2UiOiJjNDJkYjU5Ny03YWMwLTQ0NGUtYWViNS00NmUzNmMyYjEzNWEiLCJzY29wZSI6WyJhcGkiLCJvZmZsaW5lX2Fj
iQXBwbGljYXRpb24iXX0.QqcqlIrUbF2tS7rc9FfBap26Eh6T3QjaY13DH--6pbGd0cd7_U_QEG1UJAscL2iX8aZt0tcV72SWmx
VYDBoiA3wvD7-hoXzxjQ1OFUrogEG3yLz4LR3jq7mOP6ZmeRvIb52eeo5JZmntsRHbhvjxqolWS6XOOki_7aS1kK9rAK5efwHcA
LX0cL4SryW1lxuSAvqFzCANAbxbIFbPvHcY3kRZlXVRs8OiU1bycHa_fq0iuYA81mPc3-gQc8dEUhjMP9wyeKGd_1jVpMCExu_X
iGB_X9kZA HTTP/1.1" 101 6009 "-" "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.3
o) Chrome/123.0.0.0 Safari/537.36"
10.54.20.25 - - [14/May/2024:04:13:11 +0000] "GET /notifications/hub?access_token=eyJ0eXAiOiJKV1QiLC
```

Figure 31: Saving the complete JWT in a log file

exploitability

A JWT transported in the URL could be logged in various places, e.g. in the user's browser history or in log files of intermediate proxy servers or the web server.

If a JWT reaches an attacker via the exposed locations, it could be used to impersonate a legitimate user to the application, which could ultimately lead to the user's account being compromised.

In the context of Vaultwarden, an attacker in possession of a user's JWT could make certain changes to the user's personal and organizational settings. These include actions that do not require re-authentication with the master password hash:

- Deleting the *emergency access*
- Changing the roles of organization members

Note: If an attacker had a JWT, he could retrieve encrypted data from the user, but he would not be able to decrypt the data. Decryption requires knowledge of the user's master password. This is why finding it is only considered to be a low risk.

measure

Sensitive information should never be transported within the URL. The websocket endpoint should check the authorization of the requesting user based on the access token transported in a custom HTTP header, rather than using URL query parameters for this purpose.

E6 logic

While many security problems are due to faulty or missing security controls such as authentication or input validation, vulnerabilities in business logic abuse legitimate

Processing of an application in an unpredictable manner. The effects depend heavily on the specific application, but often include timing attacks, interrupting the intended sequence of steps, or repeatedly calling a function.

Since automated scans are usually unable to take into account the business logic of an application, this category of vulnerabilities must be tested manually.

E6.1 Denial-of-Service: IP-based user blocking possible [low]

observation

Vaultwarden prevents credential guessing by blocking the requesting IP from accessing the login endpoint for a period of time after several failed login attempts. After that, when a user tries to log in to any account from a blocked IP, the application always responds with a 429 status code.

Figure 32: Error message for IP-based user blocking

exploitability

IP-based access restriction may not work as expected if many users access the application from the same IP address. This is a common scenario for users using public proxies or for users in an organization where outbound Internet traffic goes through a single network gateway. In such a case, a brute-force attack can result in blocking the shared IP of many users, making the application unavailable to the legitimate users (denial of service).

vulnerability in the code

- Source:https://github.com/danigarcia/vaultwarden/blob/1.30.3/src/ratelimit.rs#L22-L29

The access restriction check is in the file `src/ratelimit.rs` implemented. The corresponding function is `check_limit_login`. As in As can be seen in the source code, the function only takes into account the source IP of the request. The parameters of the threshold are determined by the environment variables *LOGIN_RATELIMIT_SECONDS*and *LOGIN_RATELIMIT_MAX_BURST*By default, a block is triggered when 10 flagged requests are received from an IP address within 60 seconds.

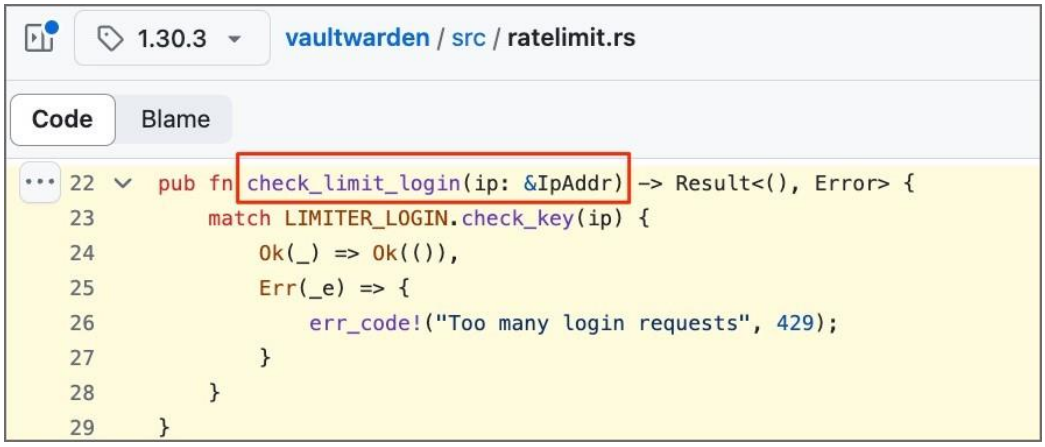


Figure 33: Denial-of-Service - vulnerability in the source code

measure

The solution is usually not to lock out the user or their IP after a certain number of failed attempts. In particular, blocking IP addresses can lock out more than one user under certain circumstances, for example if the IP address of a publicly used proxy or a proxy of a large company is blocked.

There are several measures to make DoS against user accounts more difficult for an attacker:

- Add a time delay during login after several failed login attempts ("tar pit").
- Use of classic CAPTCHAs or CAPTCHA alternatives such as "Fun CAPTCHAs" or Google's reCAPTCHA, which must be solved after a certain number of failed attempts and ask the user to make associations based on shape, colors or logical relationships.

  Detecting password cracking attempts by monitoring traffic using a web application firewall or an IDS/IPS that reacts according to defined rules and automatically initiates countermeasures.

E6.2 Insufficient anti-automation [low]

observation

Vaultwarden protects some of its API endpoints with strict rate limiting to prevent automated attacks, but the password hint request endpoint is not covered by this protection.

This allows any unauthenticated user to use an automated script to send a large number of password hint requests. This would cause the application to send the corresponding number of emails to the specified email address. As can be seen in the screenshot below, over a hundred emails were sent to a test mailbox via the application within a short period of time.

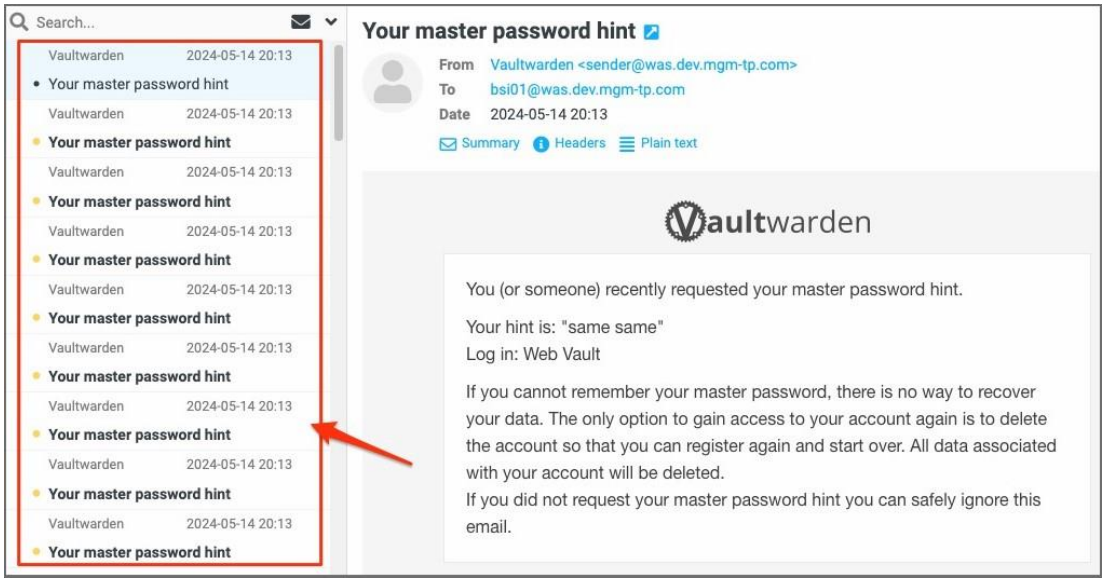


Figure 34: Email flooding via password hint

Additionally, when changing the email address of the associated account, the user must confirm that he or she is indeed the owner of the new email address. This is done by submitting a 6-digit verification code that is sent to the new email address. Vaultwarden has also not implemented rate limiting at the endpoint for submitting the verification code. Consequently, it might be possible to guess the code.

During testing, over 100 requests were sent with invalid email verification code. After that, one request with the valid code still succeeded. The successful request can be seen in the screenshot below with HTTP status code 200 for the correct verification code in request #102.

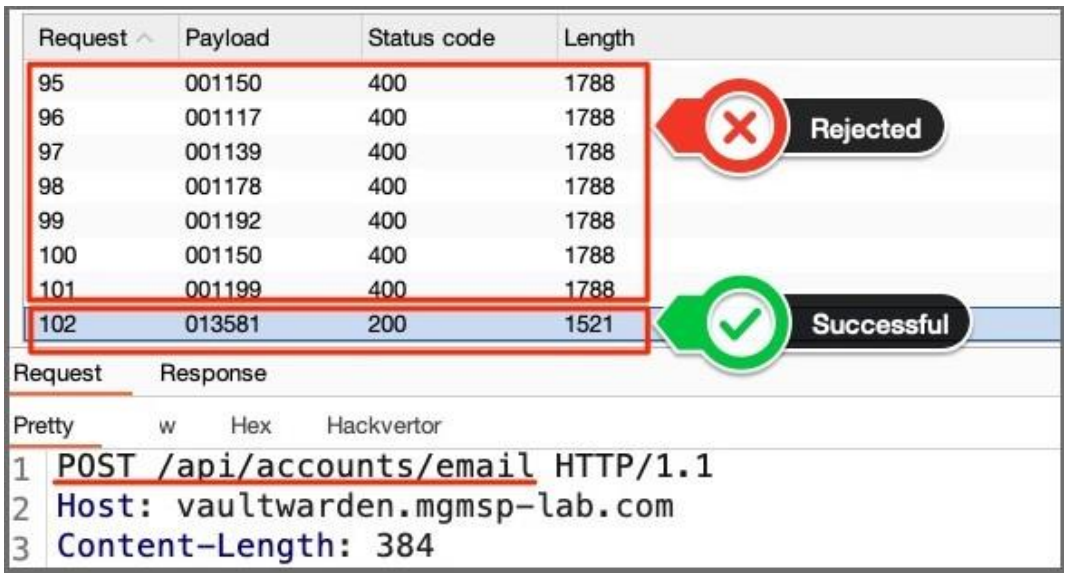


Figure 35: Successful guessing of the verification code

exploitability

Attackers could send a large number of requests to the application targeting functions that consume extensive system resources in order to conduct a denial-of-service attack. As a result, the system could become overloaded and unavailable to other users.

Another attack vector is email flooding. A comprehensive attack on the functionality for obtaining the password hint could result in the email server used being blacklisted by other email servers. Consequently, emails sent from the application could be marked as spam by some email services. Since the attack in this case can only be carried out against existing users (registered email addresses), the circle of potentially affected users is limited.

In addition, if a successful brute-force attack on the verification code is performed, an attacker can bypass the email verification step and assign an email address that does not belong to him to his own account. This could be useful, for example, in social engineering attacks.

Note: The two affected API endpoints do not require authentication.

measure

For a given user account (email address), the application should send the password hint only once within a certain time window, e.g. one day. Repeated requests for the same user account should be ignored.

The email address verification endpoint (`/api/accounts/email`)
Strict rate limiting should be applied, as already exists for other endpoints.

E6.3 Upload of any file format possible [info]

observation

Vaultwarden*Send*s is a feature that allows users to share information with others in a secure way.*Send*-Items, either text or a file, are stored on the Vaultwarden server in encrypted form and can only be decrypted with knowledge of the encryption key.

When creating a*Send*object, the user can upload any file to the server. With normal use of the*Send*-This feature encrypts the file content from the client before storing it on the Vaultwarden server. Vaultwarden allows the user to choose any file name. Consequently, files with potentially dangerous executable extensions such as

`. sh` or `. exe` be uploaded.

In the following example, a shell file (`. sh`) uploaded to the server.

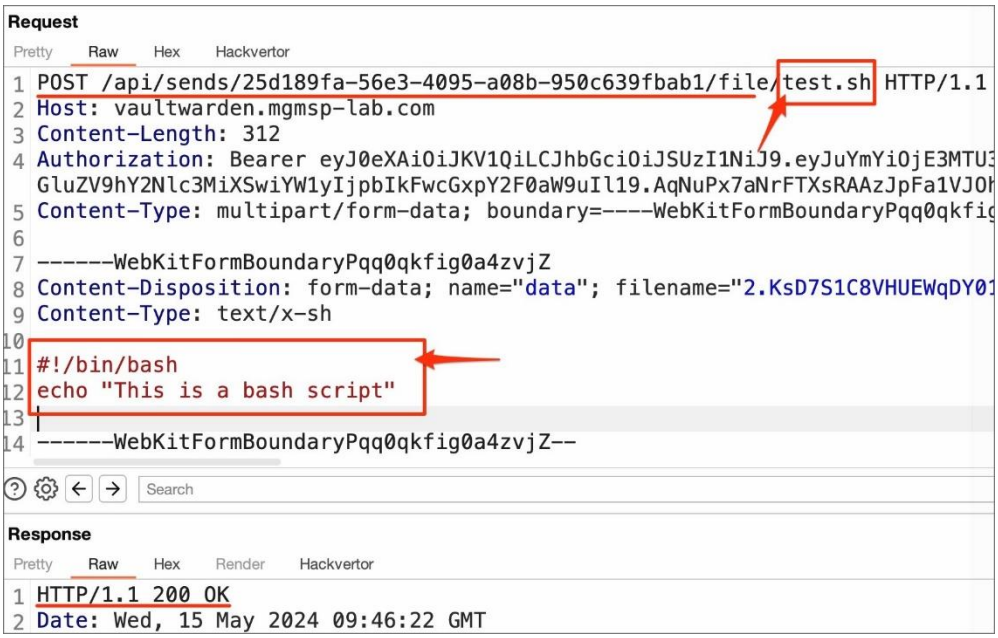


Figure 36: Uploading files with any extension.

The above request results in the test file being `test.sh` to the server uploaded.

Figure 37: Uploaded file 'test.sh' on the Vaultwarden server file system.

Although it is possible to upload a file with any file extension, there is no way to download the file again. If the file name contains a dot ( ), a download request returns an error. The reason for this behavior is that the download method does not read the request parameters (the `file_id` - parameter in the screenshot below) into a Vaultwarden-specific one. The type `SafeString` constructor of this `SafeString` - type triggers a Error if the parameter contains certain characters (only alphanumeric characters and hyphens are allowed).

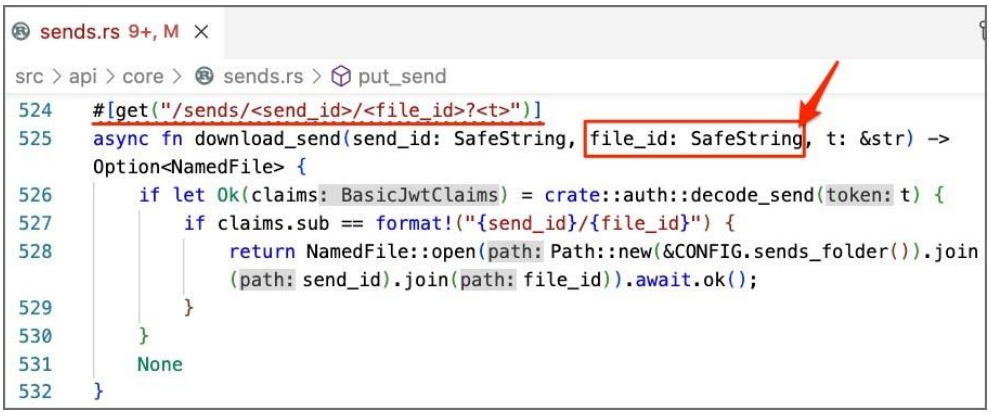


Figure 38: Files with file extension cannot be downloaded.

exploitability

An attacker could upload a file with any file format and content to the Vaultwarden server.

If the attacker succeeds in executing files on the operating system (e.g. by exploiting a *Local-File-Inclusion* vulnerability) or an authorized backend user accidentally executes the file uploaded by the attacker, then this vulnerability allows the execution of arbitrary code on the target system. In the context of Vaultwarden, however, this attack vector is rather unlikely because uploaded files are not marked as executable.

In addition, the uploaded file could be used to circumvent the *content security policy* (see Finding E3.1). The attacker could upload a JavaScript file and inject the file via a *cross-site scripting* vulnerability (see Finding E3.1) and an HTML
script - Load tag to load the CSP *script-src* directive.
However, in the context of Vaultwarden, this attack vector does not work because the download endpoint returns the file without specifying the content type and the *X-Content-Type-Options* header to the value nosniff In this
In this context, all modern browsers refuse to execute JavaScript from the
script - Tag loaded file.

Note: Since no direct exploitability could be proven during the penetration test, the finding is for information purposes only.

measure

The file upload endpoint should have the parameter file_id at the upload endpoint
The parameter should be in the custom *SafeString* type (similar to the download endpoint) to ensure that the parameter does not contain restricted and potentially malicious characters.

E7 disclosure of information

Information leakage is the unintentional disclosure of confidential information by a system or application. The data thus disclosed may include information about users, business data or technical information.

Often, the leak of technical information can be abused by attackers to carry out further attacks. In the first phase of an attack, as much information as possible is usually collected about the target system. The more extensive and detailed information can be obtained, the more precise attacks can be carried out. This increases the
The probability of an attack being successful is significant. Systems and applications should therefore be configured so that only the necessary information is provided. Any superfluous information should be withheld where possible.

E7.1 disclosure of information [low]

observation

The response from the API endpoint `/api/config` gives the exact version of the the Vaultwarden server used.

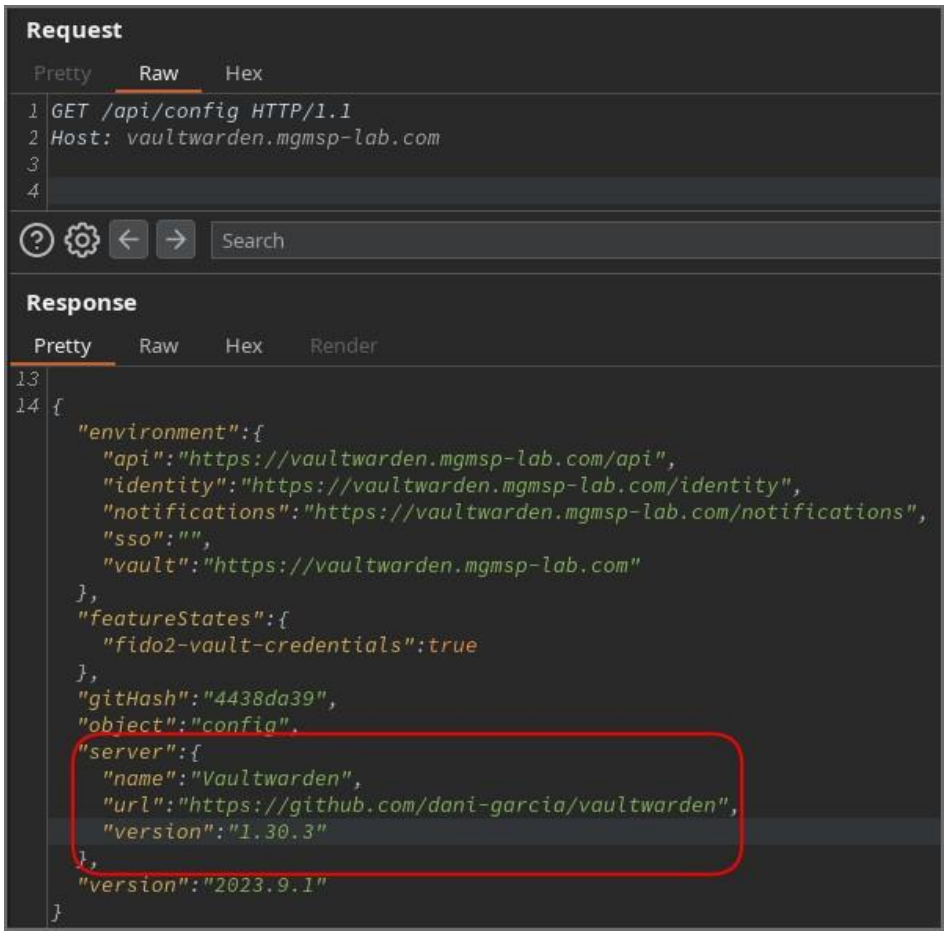


Figure 39: Disclosure of the version number

exploitability

An attacker could use the internal information obtained to plan further attacks on the server or application. They could also look for known vulnerabilities and exploits that may be publicly available.

measure

No internal information should be disclosed in the server response. In particular, no additional sensitive information such as version strings or the software used should be transmitted.

E8 session management

Session management generally describes the technical and logical requirements necessary to operate sessions, actions and transactions in an environment with stateless connections.

E8.1 Insufficient cookie configuration in the admin dashboard [low]

observation

Vaultwarden comes with an admin dashboard. This functionality can be enabled as described in the instructions linked below.

https://github.com/dani-garcia/vaultwarden/wiki/Enabling-admin-page

After activation, the Vaultwarden Admin Dashboard is located under the endpoint `/admin` In contrast to the actual application, where logged in users have to log in with a `Authorization` - header transmitted JWT authenticate, use the admin dashboard*cookies*as a transport medium for the JWT.

As you can see in the following screenshot, the cookie is set without `Secure` - Flag this.

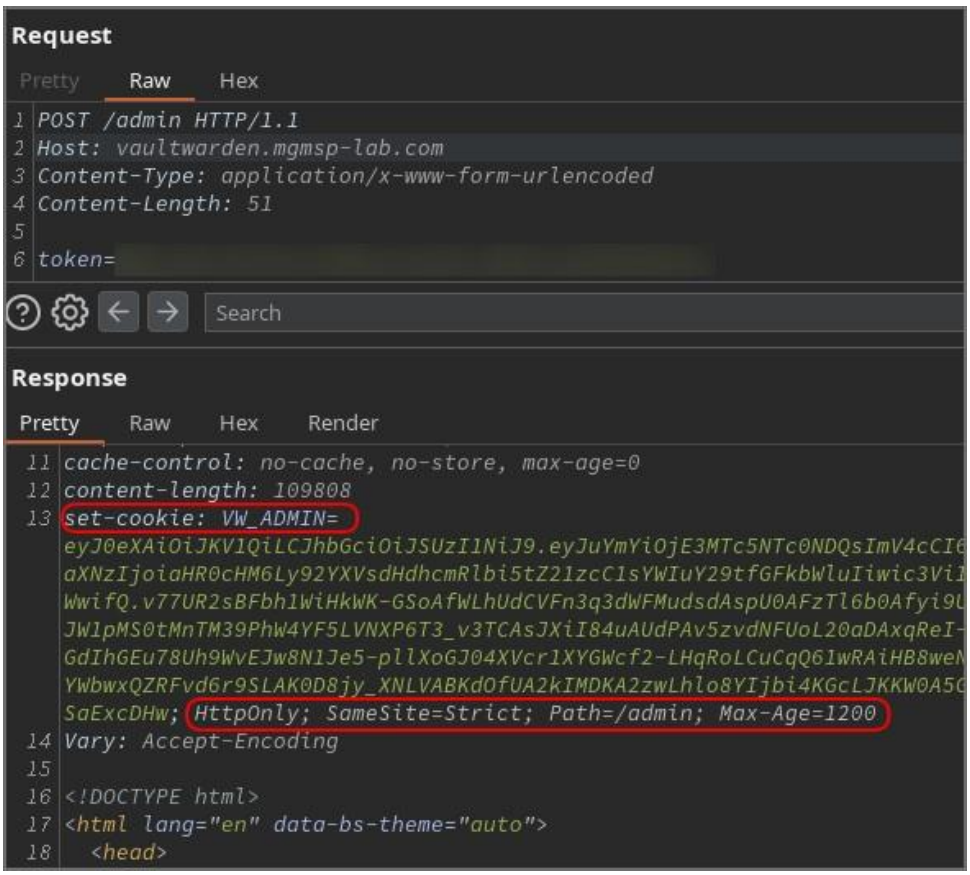


Figure 40: VW_ADMIN session cookie is set without the secure flag

It should be noted that the Vaultwarden server itself does not know whether it was accessed via HTTP or HTTPS in order to `Secure` - Cookie attribute

to enable accordingly. Typically, in a normal production deployment, the application always listens on a clear text HTTP port. HTTPS could be added by a reverse proxy sitting in front of the Vaultwarden server. In such a context, the Vaultwarden server relies on the reverse proxy to determine the connection protocol. The web server configuration examples on Vaultwarden Proxy Examples already include the mechanism for passing the connection protocol to the Vaultwarden server (using the custom header `X-Forwarded-Proto`).

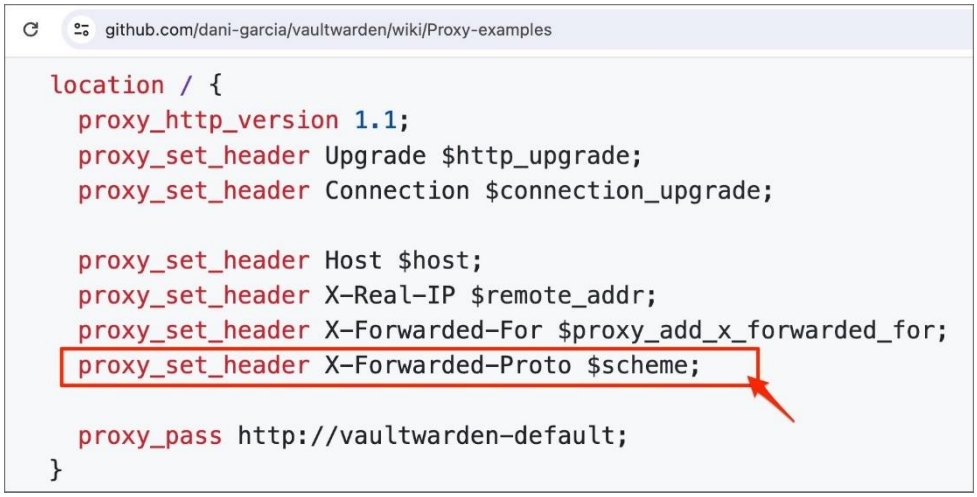


Figure 41: Example nginx configuration

exploitability

An insecure cookie configuration facilitates the compromise of a user's session.

The admin cookie `VW_ADMIN` is without the attribute `Secure` transmitted. Thus, an unencrypted transfer of the cookies back to the server is permitted. By using the attribute `Secure` the client's browser instructed to transmit the cookies to the server exclusively via encrypted connections. This makes it significantly more difficult to read the cookies in man-in-the-middle attacks.

measure

To ensure that the secure flag for the session cookie is always enabled when the user connects via HTTPS, the Vaultwarden server should `X-forwarded proto` - Check header in the request. If this header is `https` is set, the Secure flag should be added.

E9 data validation

Most vulnerabilities in web applications are caused by insufficient data validation (often referred to as input data validation).

Data validation refers to all data and associated checks that process the data that comes into a system from outside. This includes not only the direct user input, but also the data that comes from third-party systems (e.g. from databases). Furthermore, every system must ensure that the data that is passed on to third-party systems for processing does not contain any malicious code.

It is therefore important to distinguish between input validation and output encoding. This difference will be highlighted where appropriate in the following individual vulnerabilities and threats.

E9.1 log injection [low]

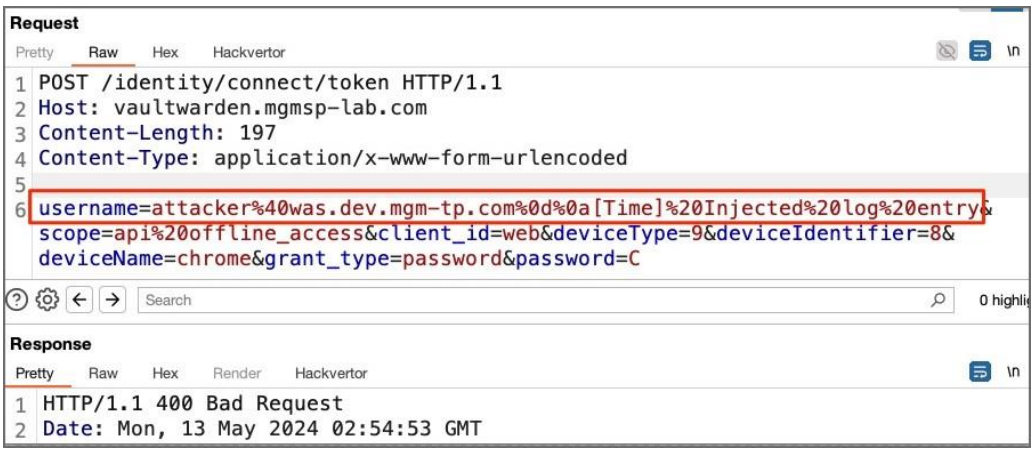
observation

By default, Vaultwarden logs certain user-triggered events in a log file located on the server. When using the official Docker image, the corresponding log file is located at `/data/vaultwarden.log` and uploaded to the host via a Docker volume. computer. More information about Vaultwarden logging can be found at the following link:

https://github.com/dani-garcia/vaultwarden/wiki/Logging

Line breaks are not correctly encoded when logging user-controlled input. A line break (`%0d%0a`) in the user input thus leads to the log file to a new line. This allows an attacker to falsify log entries or inject malicious content into the file.

In the following example, an invalid login request was sent to the Vaultwarden server. This request contains a line break in the `username` .



The above request is logged in the log file as follows. The line after the logged user name gives the impression of being another entry in the log file due to the line break. The fake log entry would be indistinguishable from a real one.

```
     @BSI-CAOS3:/var/www/vw-data$ tail -n 16 vaultwarden.log
[2024-05-13 02:54:53.458][request][INFO] POST /identity/connect/token
[2024-05-13 02:54:53.461][vaultwarden::api::identity][ERROR] Username or password is incorrec
t. Try again. IP: 10.54.20.25. Username: attacker@was.dev.mgm-tp.com
[Time] Injected log entry.
[2024-05-13 02:54:53.463][response][INFO] (login) POST /identity/connect/token => 400 Bad Req
uest
```

exploitability

This vulnerability allows an attacker to insert arbitrary content into the Vaultwarden server log file. This could be used, for example, to try to cover the tracks of an attack or even attribute the attack to another party.

measure

Special characters contained in user-controlled input and included in log files should be correctly encoded before storage. In the case of Vaultwarden, log entries are separated by a line break character. Therefore, especially *Newline* characters in logged inputs. For example, with URL encoding to `%0d%0a`.

F. Appendices

The following appendices were provided as part of this report:

- Folder: burp

  -work protocols Burp analyses

- vaultwarden-server-audited.xlsx, vaultwarden-browser-extension-audited.xlsx

  - Evaluates results of all used SAST/SECRETS scanners in an Excel document (see section D1)

-vaultwarden-server-excluded.xlsx, vaultwarden-browser-extension-excluded.xlsx

  - Excel file with a list of all SAST/SECRETS findings excluded from the assessment (see section D1)

-vaultwarden-server-sca.xlsx

  - Deduplicated results of all SCA scanners used in an Excel document (see section D2)

-vaultwarden-server-sca-excluded.xlsx

  - Findings excluded as duplicates from all SCA scanners in an Excel document (see section D2)

-Folder: raw

  - Unevaluated raw data of all scanners used (see section D)

G. References

/1/ Input and output data validation in web applications https://cwe.mitre.org/data/definitions/79.html

/2/ Overview of data validation https://www.owasp.org/index.php/Data_Validation

/3/ Cheat Sheet for data validation https://www.owasp.org/index.php/Input_Validation_Cheat_Sheet

/4/ Overview of XSS prevention https://www.owasp.org/index.php/XSS_%28Cross_Site_Scripting%29_Prevention_Ch eat_Sheet

/5/ Unvalidated Redirects and Forwards https://www.owasp.org/index.php/Unvalidated_Redirects_and_Forwards_Cheat_Sheet

/6/ Overview of SQL Injection https://www.owasp.org/index.php/SQL_Injection

/7/ Overview of Blind SQL Injection https://www.owasp.org/index.php/Blind_SQL_Injection

/8/ SQL Injection Prevention https://www.owasp.org/index.php/SQL_Injection_Prevention_Cheat_Sheet

/9/ Best practice measures for auditing and hardening databases https://benchmarks.cisecurity.org/downloads/browse/index.cfm?category=benchmarks.servers.database

/10/ Call system commands https://cwe.mitre.org/data/definitions/78.html

/11/ path traversal https://www.owasp.org/index.php/Path_Traversal

/12/ Buffer overflow attack https://www.owasp.org/index.php/Buffer_overflow_attack

/13/ format string attack https://www.owasp.org/index.php/Format_string_attack

/14/ user enumeration https://www.owasp.org/index.php/Testing_for_user_enumeration_(OWASP-AT-002)

/15/ OWASP - Blocking Brute Force Attacks https://www.owasp.org/index.php/Blocking_Brute_Force_Attacks

/16/ Length and complexity of passwords https://www.owasp.org/index.php/Authentication_Cheat_Sheet#Password_Complexity

/17/ Overview of Authentication https://www.owasp.org/index.php/Authentication_Cheat_Sheet

/18/ Overview of session management https://www.owasp.org/index.php/Session_Management_Cheat_Sheet

/19/ Session fixation vulnerability http://www.acros.si/papers/session_fixation.pdf

/20/ Introduction to CSRF or Session Riding https://www.owasp.org/index.php/Cross-Site_Request_Forgery_(CSRF)

/21/ CSRF prevention https://www.owasp.org/index.php/Cross-Site_Request_Forgery_%28CSRF%29_Prevention_Cheat_Sheet

/22/ expansion of privileges https://cwe.mitre.org/data/definitions/269.html

/23/ access control https://www.owasp.org/index.php/Access_Control_Cheat_Sheet

/24/ Secure password management https://www.owasp.org/index.php/Password_Storage_Cheat_Sheet

/25/ Secure data storage https://www.owasp.org/index.php/Cryptographic_Storage_Cheat_Sheet

/26/ Best Practices for a Secure “Forgot Password” Feature https://www.owasp.org/index.php/Forgot_Password_Cheat_Sheet

/27/ Broken user authentication and session management https://www.owasp.org/index.php/Broken_Authentication_and_Session_Management

/28/ Denial of Service https://www.owasp.org/index.php/Denial_of_Service

/29/ CAPTCHA: Telling Humans and Computers Apart Automatically http://www.captcha.net/

/30/ SSL Best Practices https://www.ssllabs.com/projects/best-practices/index.html

/31/ Overview of publicly known vulnerabilities https://nvd.nist.gov/

/32/ Secure Coding of CORS http://www.andlabs.org/html5/rejectCOR.php

/33/ Information about HSTS http://blog.securenet.de/2012/11/02/ssl-stripping-die-ignorierte-gefahr/

/34/ CWE-77: Improper Neutralization of Special Elements used in a Command ('Command Injection') https://cwe.mitre.org/data/definitions/77.html

/35/ OWASP Content Security Policy https://www.owasp.org/index.php/Content_Security_Policy

/36/ OWASP Clickjacking https://www.owasp.org/index.php/Clickjacking

/37/ OWASP HTML5 Security Cheat Sheet https://www.owasp.org/index.php/HTML5_Security_Cheat_Sheet

/38/ OWASP HTTP Headers https://www.owasp.org/index.php/List_of_useful_HTTP_headers

/39/ Unrestricted Upload of File with Dangerous Type https://cwe.mitre.org/data/definitions/434.html

/40/ Unrestricted File Upload https://www.owasp.org/index.php/Unrestricted_File_Upload

/41/ OWASP TLS Cheat Sheet https://www.owasp.org/index.php/Transport_Layer_Protection_Cheat_Sheet

/42/ Secure password storage
https://paragonie.com/blog/2016/02/how-safely-store-password-in-2016