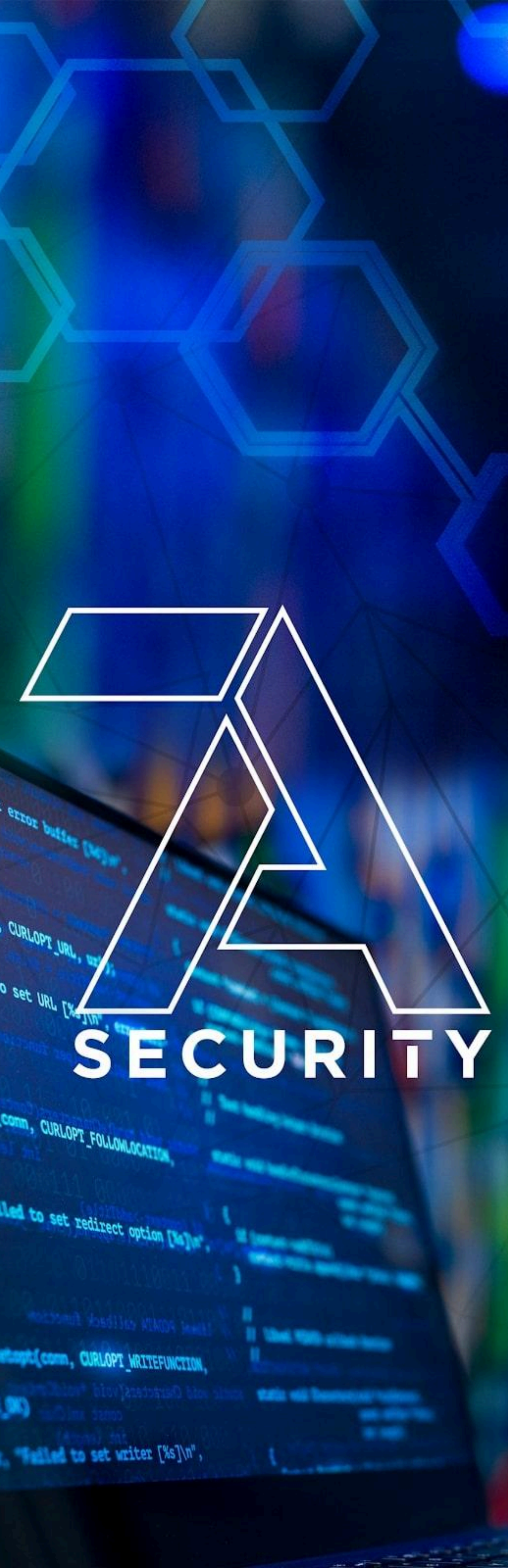


ISO/IEC 27001:2022
ISMS Certified
by Consilium Labs (IAS)

Pentest Report

Client:
eQualitie

dComms Test Targets:

Docker Orchestration
Docker Configuration
Deployment Surface
Censorship Resistance

7ASecurity Test Team:

- Abraham Aranguren, MSc.
- Dariusz Jastrzębski
- Szymon Grzybowski, MSc.
- Dheeraj Joshi, B.Tech.

7ASecurity
Protect Your Site & Apps
From Attackers
sales@7asecurity.com
7asecurity.com

INDEX

Introduction

"dComms provides decentralized communication solutions in regions where infrastructure is restricted by repression, war, or climate disasters."

From https://equalit.ie/portfolio/decentralized-communications/

This document outlines the results of a penetration test and *whitebox* security review conducted against the dComms platform. The project was solicited by eQualitie, funded by the Open Technology Fund (OTF), and executed by 7ASecurity in October and November 2025. The audit team dedicated 12.35 working days to complete this assignment. Please note that this is the first penetration test for this project. Consequently, the identification of security weaknesses was expected to be easier during this engagement, as more vulnerabilities are identified and resolved after each testing cycle.

During this iteration, the goal was to review the dComms solution as thoroughly as possible, to ensure dComms users can be provided with the best possible security. The methodology implemented was *whitebox*: 7ASecurity was provided with access to a staging environment, documentation, test users, and source code. A team of 4 senior auditors carried out all tasks required for this engagement, including preparation, delivery, documentation of findings and communication.

A number of necessary arrangements were in place by October 2025, to facilitate a straightforward commencement for 7ASecurity. In order to enable effective collaboration, information to coordinate the test was relayed through email, as well as a shared Mattermost channel. The dComms team was helpful and responsive throughout the audit, which ensured that 7ASecurity was provided with the necessary access and information at all times, thus avoiding unnecessary delays. 7ASecurity provided regular updates regarding the audit status and its interim findings during the engagement.

The findings of the security audit can be summarized as follows:

Identified Vulnerabilities	*Hardening Recommendations*	*Total Issues*
4	*6*	*10*

Moving forward, the scope section elaborates on the items under review, while the findings section documents the identified vulnerabilities followed by hardening recommendations with lower exploitation potential. Each finding includes a technical description, a proof-of-concept (PoC) and/or steps to reproduce if required, plus mitigation or fix advice for follow-up actions by the development team.

Finally, the report culminates with a conclusion providing detailed commentary, analysis, and guidance relating to the context, preparation, and general impressions gained throughout this test, as well as a summary of the perceived security posture of the dComms components.

Scope

The following list outlines the items in scope for this project:

- **WP1: Security Audit of Docker Orchestration and Configuration**
    - Source code: https://github.com/equalitie/dcomms-docker-compose
    - Documentation: https://equalit.ie/portfolio/decentralized-communications/
- **WP2: Deployment Surface and Censorship Resistance Review**
    - As above

Identified Vulnerabilities

This area of the report enumerates findings that were deemed to exhibit greater risk potential. Please note these are offered sequentially as they were uncovered, they are not sorted by significance or impact. Each finding has a unique ID (i.e. *DCO-01-001*) for ease of reference, and offers an estimated severity in brackets alongside the title.

DCO-01-004 WP2: dComms Installation Detection via CT Logs *(High)*

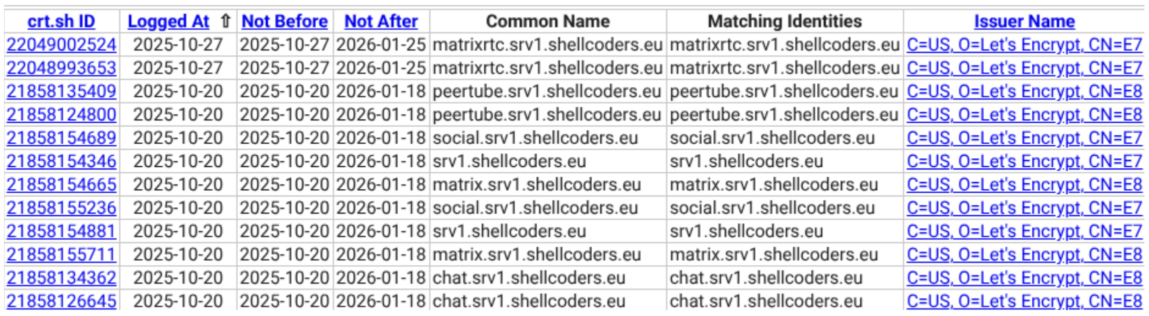
Retest Notes: Resolved by dComms[1], and verified by 7ASecurity.

It was noted that the dComms solution registers services across multiple subdomains on the server. An adversary seeking to discover installations can monitor Certificate Transparency (CT) logs and target servers and domains that register *peertube.**, *matrix.**, *social.** and *chat.** subdomains. The adversary can then pursue the server operator, VPS provider, or domain registrar to obtain personal information or configure country-wide firewalls to block IP addresses; these actions can be automated if scripted.

Certificate Transparency logs were retrieved from *crt.sh*; the logs can be used to fingerprint *dComms* installations.

PoC URL:
https://crt.sh/?q=shellcoders.eu

Output:



crt.sh ID	Logged At ⇧	Not Before	Not After	Common Name	Matching Identities	Issuer Name
22049002524	2025-10-27	2025-10-27	2026-01-25	matrixrtc.srv1.shellcoders.eu	matrixrtc.srv1.shellcoders.eu	C=US, O=Let's Encrypt, CN=E7
22048993653	2025-10-27	2025-10-27	2026-01-25	matrixrtc.srv1.shellcoders.eu	matrixrtc.srv1.shellcoders.eu	C=US, O=Let's Encrypt, CN=E7
21858135409	2025-10-20	2025-10-20	2026-01-18	peertube.srv1.shellcoders.eu	peertube.srv1.shellcoders.eu	C=US, O=Let's Encrypt, CN=E8
21858124800	2025-10-20	2025-10-20	2026-01-18	peertube.srv1.shellcoders.eu	peertube.srv1.shellcoders.eu	C=US, O=Let's Encrypt, CN=E8
21858154689	2025-10-20	2025-10-20	2026-01-18	social.srv1.shellcoders.eu	social.srv1.shellcoders.eu	C=US, O=Let's Encrypt, CN=E7
21858154346	2025-10-20	2025-10-20	2026-01-18	srv1.shellcoders.eu	srv1.shellcoders.eu	C=US, O=Let's Encrypt, CN=E7
21858154665	2025-10-20	2025-10-20	2026-01-18	matrix.srv1.shellcoders.eu	matrix.srv1.shellcoders.eu	C=US, O=Let's Encrypt, CN=E8
21858155236	2025-10-20	2025-10-20	2026-01-18	social.srv1.shellcoders.eu	social.srv1.shellcoders.eu	C=US, O=Let's Encrypt, CN=E7
21858154881	2025-10-20	2025-10-20	2026-01-18	srv1.shellcoders.eu	srv1.shellcoders.eu	C=US, O=Let's Encrypt, CN=E7
21858155711	2025-10-20	2025-10-20	2026-01-18	matrix.srv1.shellcoders.eu	matrix.srv1.shellcoders.eu	C=US, O=Let's Encrypt, CN=E8
21858134362	2025-10-20	2025-10-20	2026-01-18	chat.srv1.shellcoders.eu	chat.srv1.shellcoders.eu	C=US, O=Let's Encrypt, CN=E8
21858126645	2025-10-20	2025-10-20	2026-01-18	chat.srv1.shellcoders.eu	chat.srv1.shellcoders.eu	C=US, O=Let's Encrypt, CN=E8

Fig.: dComms installation leaks via Certificate Transparency Logs

It is recommended to use a wildcard domain by default. Subdomains should include randomized components to prevent disclosure through certificate transparency logs.

[1] https://github.com/equalitie/dcomms-docker-compose/commit/535…cfa

DCO-01-005 WP1: Lack of Signatures for Docker Images *(Medium)*

Retest Notes: Risk accepted. Responsibility shifted to the vendors of the containers.

It was observed that the dComms solution deploys a decentralized communication environment using multiple Docker images. None of these images are properly signed, which would otherwise prevent data tampering. If an attacker pushes an unsigned image to the container registry, users deploying or upgrading the stack may install a malicious image, resulting in a full environment compromise.

Affected Resources:
All Docker Hub images in use

Issue 1: Unsigned Docker Images

No community Docker images pulled from Docker Hub and used by dComms are signed. Only the Caddy image appears to include signatures.

Command (example – list signatures for Element Web image):
```
docker trust inspect --pretty vectorim/element-web
```

Output:
```
no signatures or cannot access vectorim/element-web
```

This can be compared with a signed image, such as *datadog/agent*.

Command (example – signed image):
```
docker trust inspect --pretty caddy:2.10
```

It was noted that Caddy version 2.10.2, installed during deployment, is not signed.

Output:
```
Signatures for caddy:2.10

SIGNED TAG   DIGEST                                                             SIGNERS
2.10         133b5eb7ef9d42e34756ba206b06d84f4e3eb308044e268e182c2747083f09de   (Repo
Admin)

Administrative keys for caddy:2.10
  Repository Key:
da51946634eeba45b3241ea213136b0252053937f8bbc880b0f9e87b0a95fd5f
  Root Key:     98f99d80f15b3f19b43d642bce838635fc32be0bcb25d0b03a8a463b237c40f0
```

Issue 2: Unverified Community Vendors

Docker Hub accounts publishing container artifacts (for example, *vectorim*, *matrixdotorg*, *equalitie*, *tootsuite*, *chocobozzz*) are not marked as verified. Although a verified badge does not guarantee complete security, it enables additional benefits, including vulnerability analysis. The program is available to both commercial and open-source organizations.

Some images are hosted by individual community users instead of organizational accounts, which increases the risk of compromise due to weaker access management.

It is recommended to sign[2][3] all container images to mitigate compromise risks. The dComms solution should require artifact signing for its components and host verified local copies of third-party images after validation. A suitable signing method, such as *Cosign*[4][5] or an equivalent (e.g., *Notary*[6]), should be evaluated because Docker Content Trust (DCT) is still supported but scheduled for deprecation[7].

It is advised to enable the environment variable *DOCKER_CONTENT_TRUST=1* during installation to verify image signatures or adopt a modern alternative once available. Although *Docker Hub* has not yet provided migration guidelines, Azure Container Registry will continue supporting DCT until March 31, 2028[8], while advising a transition to *Notary*. Artifact distribution platforms should be reviewed, and all available security features enabled to strengthen defenses against potential supply chain attacks[9][10]. Finally, individual user accounts hosting images should be converted into Organization Community Users to improve access control management.

[2] https://docs.docker.com/engine/security/trust/
[3] https://docs.sigstore.dev/cosign/signing/signing_with_containers/
[4] https://www.datadoghq.com/blog/container-image-signing/
[5] https://docs.gitlab.com/ci/yaml/signing_examples/
[6] https://notaryproject.dev/
[7] https://www.docker.com/blog/retiring-docker-content-trust/
[8] https://learn.microsoft.com/en-us/azure/container-registry/container-registry-content-trust-deprecation
[9] https://docs.docker.com/docker-hub/repos/manage/trusted-content/dvp-program/
[10] https://docs.docker.com/docker-hub/repos/manage/trusted-content/dsos-program/#docker-scout

DCO-01-006 WP1: Inadequate Container Network Segmentation *(High)*

Retest Notes: Resolved by dComms[11], and verified by 7ASecurity.

The dComms environment is deployed using Docker Compose with multiple vendor-maintained containers, where each service functions as an independent deployment unit. Due to this design, each unit should be isolated to prevent lateral movement in the event of a single service compromise. The dComms installation revealed that several containers share interfaces within the *compose_back* Docker network, allowing connections to databases and other services.

Affected Code:
https://github.com/equalitie/dcomms-docker-compose/
https://github.com/equalitie/dcomms-…/element.docker-compose.yml#L34-L45

Issue 1: Synapse Containers in a Shared *compose_back* Network

Multiple unrelated containers were identified within the same network, including a PostgreSQL database used by the Synapse server. Host-based authentication is configured for the database, allowing any host user to connect without a password. A compromise of any web container could therefore enable a pivot to the Synapse service.

Containers within the same network:
- *compose-auth-service-1*
- *compose-mastodon-web-1*
- *compose-synapse-1*
- *compose-mastodon-streaming-1*
- *compose-caddy-1*
- *compose-mau-1*
- *compose-peertube-back-1*
- *compose-synapse-pg-1*
- *compose-livekit-1*
- *compose-element-1*

Cross-connections for all published ports are permitted when containers reside within the same network, either through the network bridge or the host network.

Example: PeerTube Compromise Simulation

Command (Simulated RCE on PeerTube container):
```
docker exec -it compose-peertube-back-1 /bin/bash
```

[11] https://github.com/equalitie/dcomms-docker-compose/commit/013…140

Command (DNS resolution for Synapse PostgreSQL container):
```
host compose-synapse-pg-1
```

Output:
```
compose-synapse-pg-1 has address 172.19.0.6
```

Command (Connection attempt from PeerTube to PostgreSQL database):
```
psql -h 172.19.0.6 -U postgres
```

Output:
```
psql (15.14 (Debian 15.14-0+deb12u1), server 14.19)
Type "help" for help.

postgres=#
```

Command (Basic recon):
```
postgres=# \conninfo \du \dt
```

Output:
```
You are connected to database "postgres" as user "postgres" on host "172.19.0.6" at
port "5432".
                                   List of roles
 Role name |                         Attributes                         | Member of
-----------+------------------------------------------------------------+-----------
 postgres  | Superuser, Create role, Create DB, Replication, Bypass RLS | {}

                             List of relations
 Schema |                       Name                       | Type  |  Owner
--------+--------------------------------------------------+-------+----------
 public | access_tokens                                    | table | postgres
 public | account_data                                     | table | postgres
 public | account_validity                                 | table | postgres
 public | application_services_state                       | table | postgres
 public | application_services_txns                        | table | postgres
 public | applied_module_schemas                           | table | postgres
 public | applied_schema_deltas                            | table | postgres
 public | appservice_room_list                             | table | postgres
 public | appservice_stream_position                       | table | postgres
 public | background_updates                               | table | postgres
 public | blocked_rooms                                    | table | postgres
 public | cache_invalidation_stream_by_instance            | table | postgres
 public | current_state_delta_stream                       | table | postgres
 public | current_state_events                             | table | postgres
 public | dehydrated_devices                               | table | postgres
 public | delayed_events                                   | table | postgres
 public | delayed_events_stream_pos                        | table | postgres
 public | deleted_pushers                                  | table | postgres
 public | destination_rooms                                | table | postgres
 public | destinations                                      | table | postgres
 [...]
```

Successful access confirms potential for lateral movement and database compromise.

Issue 2: Lack of Internal Bridge Networks

The *internal* property is set to *false* for all networks, allowing every container to communicate with the Internet and internal networks within the hosting data center.

Command (check network properties):

```
docker network inspect compose_back
```

Output:

```
[
    {
        "Name": "compose_back",
        "Id": "d671246b8d9ac2b4ba84393e9fb827a1f6a5389179b50b2d9756c69044c636d3",
        "Created": "2025-11-04T07:38:51.938378794Z",
        "Scope": "local",
        "Driver": "bridge",
        "EnableIPv4": true,
        "EnableIPv6": false,
        "IPAM": {
            "Driver": "default",
            "Options": null,
            "Config": [
                {
                    "Subnet": "172.19.0.0/16",
                    "Gateway": "172.19.0.1"
                }
            ]
        },
        "Internal": false,
        "Attachable": false,
        "Ingress": false,
        "ConfigFrom": {
            "Network": ""
        },
[...]
    }
]
```

The output confirms that it allows unrestricted outbound communication. If a container is compromised, an attacker may attempt to access the main host network or cloud metadata endpoints if deployed in a cloud environment.

It is recommended to implement service isolation through dedicated networks to minimize the impact of potential container compromises. The reverse proxy should be configured with multiple interfaces to ensure proper routing between isolated networks. Internal[12] bridge networks should be utilized wherever feasible to prevent unnecessary exposure to external and inter-container communication paths.

DCO-01-008 WP2: Backend IP Exposure via Mastodon Link Previews *(Medium)*

Retest Notes: Risk accepted. While affecting dComms, this issue can only be fixed by Mastodon[13]. dComms could warn users about this weakness in the meantime.

The Mastodon service, a self-hosted equivalent of X (formerly Twitter) deployed as part of dComms, permits link previews that expose both the backend server IP address and the domain name of the environment. Users operating in hostile environments who post links within the instance, for example, to governmental websites, trigger outbound connections that may be monitored. These requests reveal the backend IP address and domain name, which is particularly undesirable for instances using Tor or hidden Onion services.

Affected Service (dComms Mastodon):
https://github.com/equalitie/dcomms-.../mastodon.docker-compose.yml

When a user posts a plain link in a comment or post, the Mastodon service initiates a server-side HTTP request to the linked resource. The *User-Agent* header of the request includes both information about the Mastodon bot and the URL of the dComms environment. This behavior enables adversaries to identify dComms installations through simple monitoring.

Example Server-Side Request:
```
GET /test.png HTTP/1.1
User-Agent: Mastodon/4.4.7 (http.rb/5.3.1; +https://social.<dComms WEBDOMAIN>/) Bot
Host: <attacker controlled host>
Date: Thu, 06 Nov 2025 07:44:18 GMT
Accept-Encoding: gzip
Accept: text/html
Accept-Language: en, *;q=0.5
Connection: close
```

It was confirmed that the Synapse service has link previews disabled by default and is therefore not affected.

[12] https://docs.docker.com/reference/cli/docker/network/create/#internal
[13] https://github.com/mastodon/mastodon/issues/20188

It is recommended to research methods to disable link previews in Mastodon to prevent backend information leakage. Currently, this capability does not appear to exist in the official software; therefore, a feature request or the development of a custom module may be required. The related issue concerning link preview functionality should be monitored for updates[14].

[14] https://github.com/mastodon/mastodon/issues/20188

Hardening Recommendations

This area of the report provides insight into less significant weaknesses that might assist adversaries in certain situations. Issues listed in this section often require another vulnerability to be exploited, need an uncommon level of access, exhibit minor risk potential on their own, and/or fail to follow information security best practices. Nevertheless, it is recommended to resolve as many of these items as possible to improve the overall security posture and protect users in edge-case scenarios.

DCO-01-001 WP1: Insufficient Automated Image Scanning *(Low)*

Docker image security scanning is a critical element of container security and should be applied to all project images. During the assessment, several Docker images were identified as containing publicly known vulnerabilities, introducing security risks. The commands and results below highlight significant vulnerabilities in outdated images.

During testing, the *Trivy*[15] tool was used with a severity filter limited to High and Critical levels. The output is reduced for brevity, as numerous packages were found to be vulnerable to various classes of issues.

Commands:
```
trivy image --severity HIGH,CRITICAL tootsuite/mastodon-streaming:v4.4
trivy image --severity HIGH,CRITICAL dock.mau.dev/maubot/maubot:v0.5.1
trivy image --severity HIGH,CRITICAL matrixdotorg/synapse:v1.140.0
trivy image --severity HIGH,CRITICAL chocobozzz/peertube:production-bookworm
trivy image --severity HIGH,CRITICAL equalitie/ceno-client:latest
```

Output:
```
tootsuite/mastodon-streaming:v4.4 (debian 12.12):
Total: 12 (HIGH: 11, CRITICAL: 1)
CVE-2025-6020, affected package: libpam-modules, issue type: directory traversal
[...]

dock.mau.dev/maubot/maubot:v0.5.1 (alpine 3.21.0):
Total: 36 (HIGH: 31, CRITICAL: 5)
CVE-2025-27516, affected package: py3-jinja2, issue type: template injection
[...]

matrixdotorg/synapse:v1.140.0 (debian 12.12):
Total: 8 (HIGH: 6, CRITICAL: 2)
CVE-2025-6020 affected package: libpam-modules, issue type: directory traversal
[...]

chocobozzz/peertube:production-bookworm (debian 12.12):
```

[15] https://github.com/aquasecurity/trivy

```
Total: 178 (HIGH: 174, CRITICAL: 4)
CVE-2025-6020 affected package: libpam-modules, issue type: directory traversal
[...]

equalitie/ceno-client:latest (debian 12.12):
Total: 5 (HIGH: 4, CRITICAL: 1)
CVE-2025-6020 affected package: libpam-modules, issue type: directory traversal
[...]
```

Although each *dComms* project should include automated image scanning in its CI/CD pipeline, it is recommended to implement this step for the entire *dComms* solution at release.

It is recommended to update the affected Docker images to the latest stable versions to mitigate these vulnerabilities. Additionally, it is recommended to integrate a Docker image security scanning tool, such as Trivy, into CI/CD pipelines[16] to continuously identify and reduce vulnerabilities. ShellCheck[17] may also be integrated as a Static Analysis Security Testing tool for Bash scripts.

DCO-01-002 WP1: Container Images Missing Docker Digest *(Low)*

The dComms system uses the Docker ecosystem to deploy a self-hosted decentralized communications environment but does not adequately protect against container tampering. The project primarily relies on image tags (e.g., *latest* or version labels), which are mutable. An attacker capable of pushing a malicious image to the registry could compromise multiple installations, as affected systems may automatically pull the *latest* image during restart or installation, leading to a full system compromise.

By verifying images and using immutable Docker digest[18] pins, developers can ensure that end users consistently pull the same image, reducing the potential impact of a supply chain attack on any deployment component.

Command:
```
grep "image: "
```

Output:
```
compose/mastodon.docker-compose.yml:    image: redis:7.0-alpine
compose/mastodon.docker-compose.yml:    image: tootsuite/mastodon:v4.4
compose/mastodon.docker-compose.yml:    image: tootsuite/mastodon-streaming:v4.4
compose/mastodon.docker-compose.yml:    image: tootsuite/mastodon:v4.4
compose/mastodon.docker-compose.yml:    image: postgres:14.13-alpine
compose/element.docker-compose.yml:     image: matrixdotorg/synapse:v1.140.0
```

[16] https://trivy.dev/v0.41/ecosystem/cicd/
[17] https://github.com/koalaman/shellcheck
[18] https://docs.docker.com/dhi/core-concepts/digests/

```
compose/element.docker-compose.yml:     image: postgres:14-alpine
compose/element.docker-compose.yml:     image: vectorim/element-web:v1.11.110
compose/element.docker-compose.yml:#    image: matrixdotorg/mjolnir:latest
compose/element.docker-compose.yml:     image: ghcr.io/element-hq/lk-jwt-service:latest
compose/element.docker-compose.yml:     image: livekit/livekit-server:latest
compose/bridge.docker-compose.yml:      image: equalitie/ceno-client:latest
compose/peertube.docker-compose.yml:    image: chocobozzz/peertube:production-bookworm
compose/peertube.docker-compose.yml:    image: postgres:13-alpine
compose/peertube.docker-compose.yml:    image: redis:6-alpine
compose/mau.docker-compose.yml:         image: dock.mau.dev/maubot/maubot:v0.5.1
compose/docker-compose.yml:             image: caddy:2.10.2
```

It is recommended to use Docker digests for all system components and implement an atomic upgrade procedure to allow administrators to perform seamless upgrades across all components.

DCO-01-003 WP1: Unsigned Git Commits *(Medium)*

Multiple unsigned commits were identified in the Git repositories. If a developer machine is compromised or an attacker gains access to a developer SSH key, malicious code could be committed to the repository. This risk increases if additional security controls, such as mandatory approvals, fail. Signature verification should therefore be included in the code review workflow. This was confirmed as follows:

Command

```
git show 05afb87ba6897c037fec884adccf458bf4a8a0e2 --show-signature -q
```

Output

```
# Missing signature part
commit 05afb87ba6897c037fec884adccf458bf4a8a0e2 (HEAD -> main, origin/main,
origin/HEAD)
Author: A <a@a.a>
Date:   Thu Oct 16 08:43:45 2025 -0700

    Version bumps, Deltachat removal
```

The following example shows a commit that contains a signature, but the corresponding public key is not available locally. While further verification is required, it demonstrates a signed commit.

Command

```
git show 42e714b441fa2242a77c8e814649f27129a3c649 --show-signature -q
```

Output:

```
commit 42e714b441fa2242a77c8e814649f27129a3c649
gpg: Signature made Tue 08 Jul 2025 01:47:45 PM EDT
```

```
gpg:                using RSA key B5690EEEBB952194
gpg: Can't check signature: No public key
Author: A <32370941+aphick@users.noreply.github.com>
Date:   Tue Jul 8 10:47:45 2025 -0700

    Update Caddyfile.tmpl
```

Several commits in the repository were found to be unsigned, primarily by the local user *A <a.a>*.

It is recommended to sign[19] all commits to enable verification of authorship and to prevent acceptance of unsigned changes.

DCO-01-007 WP1: Unused Ouinet Component *(Info)*

The dComms solution deploys multiple Docker containers to create an environment intended to support communication in hostile regions. Among these components is a Ceno-Client Docker container, designed to operate as a node within the Ouinet network. However, this container does not expose any ports, is not configured as a proxy, and is not bridged to other services. It is launched with an incorrect port exposed and runs on the server by design. Because it is not executed locally with properly exposed ports, peer-to-peer connections cannot be established, rendering the component non-functional.

Command (list ceno-client container):

```
docker ps -a | grep ceno-client
```

Output:

```
6f8a0d68e86b   equalitie/ceno-client:latest
               "/opt/ouinet/ouinet \u2026"   10 days ago   Up 10 days
               0.0.0.0:28729->28729/tcp, [::]:28729->28729/tcp
               compose-bridge-1
```

Command (list listening ports inside the container):

```
docker exec -it compose-bridge-1 /bin/netstat -nlptu
```

Output:

```
Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address     Foreign Address     State       PID/Program name
tcp        0      0 127.0.0.1:8078    0.0.0.0:*           LISTEN      1/client
tcp        0      0 127.0.0.1:8077    0.0.0.0:*           LISTEN      1/client
tcp        0      0 127.0.0.11:42377 0.0.0.0:*           LISTEN      -
udp        0      0 0.0.0.0:35069     0.0.0.0:*                       1/client
udp        0      0 0.0.0.0:37391     0.0.0.0:*                       1/client
```

[19] https://docs.github.com/en/authentication/managing-commit-signature-verification/signing-commits

```
udp        0      0 127.0.0.11:49762 0.0.0.0:*                         -
udp        0      0 0.0.0.0:37742    0.0.0.0:*                         1/client
```

The container configuration indicates that it is bound only to localhost addresses, preventing external access and peer connectivity.

It is recommended to remove the Ceno-Client container, as its current configuration prevents use within the dComms environment. Reconfiguring it as a shared proxy could theoretically enable limited functionality in restricted environments; however, this approach is not advised due to the requirement for end-users to trust the dComms server operator with TLS certificate imports. Each user should instead run an independent Ceno-Client (Ouinet Client) instance as a personal browser proxy or operate it in bridge mode within local networks for secure and decentralized connectivity.

DCO-01-009 WP1: Insecure Default Credentials via Installer *(Low)*

It was observed during the installation of the dComms software that default credentials were retained, creating a security risk. Administrators often overlook changing these settings, leaving systems exposed to unauthorized access and possible data compromise.

Affected Files:
dcomms-docker-compose/conf/peertube/environment
dcomms-docker-compose/conf/mastodon/env.production
dcomms-docker-compose/conf/compose/element.docker-compose.yml
dcomms-docker-compose/conf/synapse/livekit.yaml
dcomms-docker-compose/conf/mjolnir/production.yaml

Issue 1: PostgreSQL Without Password

Command:
```
grep -i postgres_pass dcomms-docker-compose/conf/peertube/environment
```

Output (empty variable):
```
POSTGRES_PASSWORD=
```

The PostgreSQL database password was not set.

Issue 2: Redis Without Authentication

Command:
```
grep -i redis_pass dcomms-docker-compose/conf/mastodon/env.production
```

Output (empty variable):

```
REDIS_PASSWORD=
```

The Redis password was not configured.

Issue 3: LiveKit with Default Secret

Command:

```
grep -R LIVEKIT_SECRET dcomms-docker-compose/conf/*
```

Output:

```
devkey: "#CHANGEMENOW" #must be same as LIVEKIT_SECRET= in element.docker-compose.yml
LIVEKIT_SECRET=#CHANGEMENOW # must be same as devkey in conf/synapse/livekit.yml
```

A known default secret was used in LiveKit.

Issue 4: Mjolnir Module Weak Default Credentials

Command:

```
grep -i pass dcomms-docker-compose/conf/mjolnir/production.yaml
```

Output:

```
password: "password"
```

The Mjolnir module, despite being optional, also uses a weak default password.

It is recommended to generate random passwords during installation or require users to define custom strong credentials. All services, whether mandatory or optional, should maintain strong passwords for every account, including internal ones, to reduce exposure if a service is publicly accessible or if an attacker moves laterally after gaining access.

DCO-01-010 WP1: Insufficient Docker Container Hardening *(Low)*

It was found that dComms Docker containers execute commands without explicit user specification. By default, the containers operate as the root user, which increases the risk of privilege escalation and potential host compromise if a vulnerability is exploited.

Command (containers running as root):

```
docker exec -it <image> whoami
```

Output:

```
IMAGE:  compose-mastodon-db-1 USER: root
IMAGE:  compose-mastodon-redis-1 USER: root
IMAGE:  compose-mau-1 USER: root
IMAGE:  compose-peertube-back-1 USER: root
IMAGE:  compose-livekit-1 USER: root
IMAGE:  compose-synapse-1 USER: root
IMAGE:  compose-peertube-postgres-1 USER: root
IMAGE:  compose-bridge-1 USER: root
IMAGE:  compose-synapse-pg-1 USER: root
IMAGE:  compose-peertube-redis-1 USER: root
IMAGE:  compose-caddy-1 USER: root
```

It is recommended to create a dedicated non-root user and configure each container to run under this user. The assigned user should be granted only the minimum permissions required for proper operation to reduce the risk of privilege escalation.

Conclusion

Despite the number of findings encountered in this exercise, the dComms components defended themselves well against a broad range of attack vectors. The solution will become increasingly difficult to attack as additional cycles of security testing and subsequent hardening continue.

The dComms platform provided a number of positive impressions during this assignment that must be mentioned here:

- The deployed projects were found to be mature and well-structured, reflecting a stable and thoughtfully developed solution.
- The setup process was streamlined and intuitive, allowing non-technical users to deploy environments efficiently.
- The deployment was not easily discoverable from the public internet, adding an extra layer of protection against casual reconnaissance, aside from limited visibility through certificate transparency data (DCO-01-004).
- The documentation was comprehensive and precise, fully supporting environment setup without the need for clarification.
- The dComms platform demonstrated a strong network security posture, showing no unintended exposure of services or sensitive resources during testing.

The security of the dComms platform will improve with a focus on the following areas:

- **Service Isolation and Segmentation:** Each service should be properly isolated from the others to minimize the impact of a potential compromise. The architecture should assume that any service may be breached and ensure that it cannot affect the rest of the environment (DCO-01-006).
- **Privacy, Anonymity and Signature Verification:** Greater emphasis should be placed on privacy, data protection and prevention of server fingerprinting and discovery (DCO-01-004). The services currently used within the platform, such as PeerTube and Mastodon, are not inherently privacy-centric (DCO-01-008). The orchestration layer should enforce privacy-protective defaults and encourage the use of privacy-focused configurations, as well as enforce signature verification for all containers in use (DCO-01-005).
- **Secure Secret Management:** Default or empty deployment secrets were identified in certain configurations (DCO-01-009). These should be replaced with securely generated values to prevent unauthorized access and improve the overall security posture of the platform.
- **CI/CD Pipeline Hardening:** The CI/CD practices should be more clearly defined and strictly enforced. Container scanning tools should be integrated into the pipeline to identify vulnerable images before deployment (DCO-01-001). Strengthening security checks and quality gates would improve overall build integrity.

- **Container Privilege Management:** Several container images were found to run under the root user, which may expose the host system to unnecessary risk (DCO-01-010). Implementing least-privilege execution across all containers would significantly reduce this exposure.

It is advised to address all issues identified in this report, including informational and low severity tickets where possible. This will not just strengthen the security posture of the application significantly, but also reduce the number of tickets in future audits.

Once all issues in this report are addressed and verified, a more thorough review, ideally including another source code audit, is highly recommended to ensure adequate security coverage of the platform.

Please note that future audits should ideally allow for a greater budget so that test teams are able to deep dive into more complex attack scenarios. Some examples of this could be third party integrations, complex features that require to exercise all the application logic for full visibility, authentication flows, challenge-response mechanisms implemented, subtle vulnerabilities, logic bugs and complex vulnerabilities derived from the inner workings of dependencies in the context of the application. Additionally, the scope could perhaps be extended to include other internet-facing dComms resources.

It is suggested to test the components regularly, at least once a year or when substantial changes are going to be deployed, to make sure new features do not introduce undesired security vulnerabilities. This proven strategy will reduce the number of security issues consistently and make the application highly resilient against online attacks over time.

7ASecurity would like to take this opportunity to sincerely thank the dComms team, for their exemplary assistance and support throughout this audit. Last but not least, appreciation must be extended to the Open Technology Fund (OTF) for sponsoring this project.