Security Evaluation Report

Matridge Developers

V 1.0
Amsterdam, July 6th, 2026
Public

Document Properties

Client	Matridge Developers
Title	Security Evaluation Report
Targets	• Matridge v0.3.3 (https://codeberg.org/slidge/matridge) • Slidge v0.3.11 (https://codeberg.org/slidge/slidge)
Version	1.0
Pentester	Robert Gawlik
Authors	Robert Gawlik, Marcus Bointon
Reviewed by	Marcus Bointon
Approved by	Melanie Rieback

Version control

Version	Date	Author	Description
0.1	June 26th, 2026	Robert Gawlik	Initial draft
0.2	June 30th, 2026	Marcus Bointon	Review
1.0	July 6th, 2026	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

Radically Open Security B.V. is registered at the trade register of the Dutch chamber of commerce under number 60628081.

Table of Contents

1 Executive Summary

1.1 Introduction

Between June 8, 2026 and June 26, 2026, Radically Open Security B.V. carried out a security review of Matridge XMPP gateway, a python based framework acting as a XMPP component and Matrix client. It allows users to bridge connections from XMPP clients to Matrix communication services.

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

1.2 Scope of Work

The scope of the security audit was limited to the following targets:

- Matridge v0.3.3 (https://codeberg.org/slidge/matridge)
- Slidge v0.3.11 (https://codeberg.org/slidge/slidge)

The following areas are out-of-scope:

- XMPP server software (Prosody, ejabberd)
- Matrix homeserver software (Synapse, Dendrite)
- Third-party libraries (slixmpp, matrix-nio, aiohttp)

The security audit was carried out from a crystal-box perspective with full access to source code.

The scoped services are broken down as follows:

- Live system setup and limited code review: 1 days
- Attack surface and user-controlled data flow enumeration : 1 days
- Automated code scanning and manual tracing: 1 days
- Analysis and reproduction of potential vulnerabilities: 1 days
- Investigating user-impersonation and federation attacks: 1 days
- Documentation and report of vulnerabilities and its practical impact: 1 days
- **Total effort: 6 days**

1.3 Project Objectives

ROS will perform a penetration test of Matridge and Slidge in order to assess its security. To do so ROS will access Matridge version v0.3.3 and Slidge version v0.3.11 and attempt to find vulnerabilities, exploiting any such found to try and gain further access and elevated privileges.

We set up a live system for dynamic analysis consisting of the Prosody XMPP server with Matridge configured as an XMPP component. The focus is on static analysis with automated code scanning tools to eliminate known classes of vulnerabilities (CWEs) and weaknesses in dependencies. We will manually review user registration and authentication data flows as well as file attachment processes for potential issues.

1.4 Timeline

The security audit took place between June 8, 2026 and June 26, 2026.

1.5 Results In A Nutshell

During this crystal-box penetration test we found 3 Moderate-severity issues.

The findings relate to unencrypted credentials CLN-001 (page 14), uncaught exceptions CLN-002 (page 15), and outdated dependencies CLN-005 (page 16).

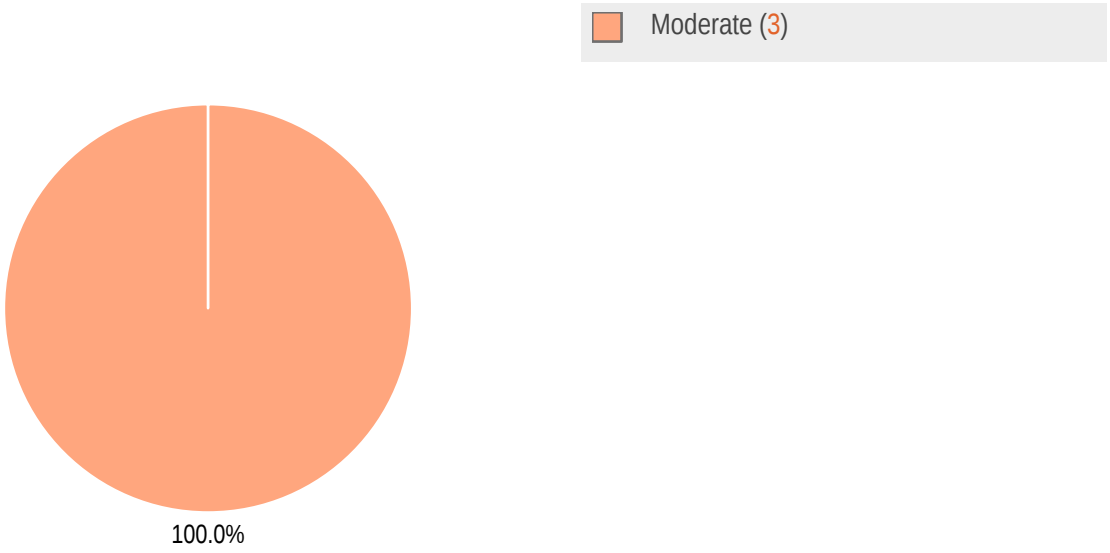
By exploiting these issues, an attacker might be able to crash the Matridge python process, requiring a manual restart, take over Matrix accounts in specific cases, or exploit weaknesses in dependencies.

1.6 Summary of Findings

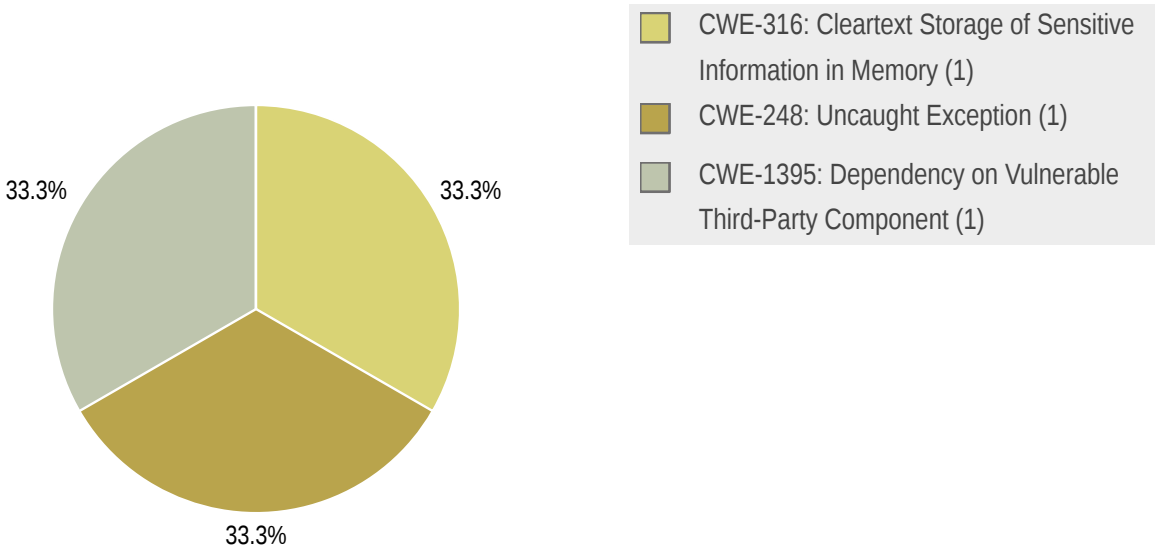
Info	Description
CLN-001 **Moderate** **Type:** CWE-316: Cleartext Storage of Sensitive Information in Memory **Status:** none	When a user registers with the gateway, the Matrix username and password are stored in plaintext in the Matridge python process memory. Cleartext credentials are also stored in a SQLite DB.
CLN-002 **Moderate** **Type:** CWE-248: Uncaught Exception **Status:** none	An authenticated XMPP user can crash Matridge when registering to the gateway.

CLN-005 **Moderate** **Type:** CWE-1395: Dependency on Vulnerable Third-Party Component **Status:** none	pip-audit reveals outdated dependencies.

1.6.1 Findings by Threat Level



1.6.2 Findings by Type



1.7 Summary of Recommendations

Info	Recommendation
CLN-001 **Moderate** **Type:** CWE-316: Cleartext Storage of Sensitive Information in Memory **Status:** none	• Encrypt credentials before storing them to disk using cryptographic primitives such as Fernet.
CLN-002 **Moderate** **Type:** CWE-248: Uncaught Exception **Status:** none	• Catch the exception with a `try except` statement wrapped around `self.discovery_info()` and handle it gracefully.
CLN-005 **Moderate** **Type:** CWE-1395: Dependency on Vulnerable Third-Party Component **Status:** none	• Integrate `pip-audit` into CI and keep packages up to date so that security patches are installed as soon as vulnerabilities in dependencies are reported and fixed.

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 Reconnaissance and Fingerprinting

We were able to gain information about the software and infrastructure through the following automated scans. Any relevant scan output will be referred to in the findings.

- Bandit – https://bandit.readthedocs.io/en/latest/
- pip-audit – https://pypi.org/project/pip-audit

We used the ipdb debugger (https://pypi.org/project/ipdb) for manual dynamic analysis to e.g., trace user-controlled input in a live setup.

3.1 Live Setup

To be able to trigger and reproduce vulnerabilities we set up a live system with default configurations consisting of Prosody (XMPP server), Matridge (component) and test users.

Prosody

After installing Prosody, we configure Matridge as a component in `/etc/prosody/prosody.cfg.lua`:

```
VirtualHost "target-example.com"
Component "matrix.target-example.com"
    component_secret = "matridge-secret"
```

We add the domain `target-example.com` to the client's `/etc/hosts` file to resolve to Prosody's IP.

Prosody accepts the following connection types:

- `target-example.com:5222` – client-to-server (c2s) – from XMPP clients
- `localhost:5347` from components such as Matridge
- `target-example.com:5269` – server-to-server (s2s) – from other XMPP servers (federation)

A self-signed certificate is necessary for XMPP clients (e.g., Gajim) to connect to `target-example.com` without errors:

```
prosodyctl cert generate target-example.com
cp /var/lib/prosody/target-example.com.crt /etc/prosody/certs/
cp /var/lib/prosody/target-example.com.key /etc/prosody/certs/
chmod 640 /etc/prosody/certs/target-example.com.{key,crt}
chown root:prosody /etc/prosody/certs/target-example.com.{key,crt}
prosodyctl check
```

Note that the certificate needs to be added to the local trust store for connections to avoid any hiccups (e.g., on ArchLinux). Similarly, we used the `prosodyctl cert` command to create certificates for other domains to be able to establish connections for federation

Matridge

We set up Matridge on the server machine in live editable mode:

```
git clone https://codeberg.org/slidge/slidge && cd slidge && git checkout v0.3.11 && cd ..
git clone https://codeberg.org/slidge/matridge && cd matridge && git checkout v0.3.3 && cd ..
git clone https://codeberg.org/poezio/slixmpp && cd slixmpp && git checkout slix-1.14.1 && cd ..
cd matridge
uv sync --frozen --all-groups --all-extras
source .venv/bin/activate
uv pip install -e ../slidge
uv pip install -e ../slixmpp
```

With the command `matridge --jid matrix.target-example.com --secret matridge-secret --home-dir=/tmp/homedir`, Matridge connects to Prosody as a component.

XMPP test users

We add the user to Prosody `testuser@target-example.com` using `prosodyctl register testuser target-example.com testpassword`. It is then able to use the Matrige gateway as bridge to Matrix chats.

3.2 Threat model

We assume two threat models:

- A registered user interacting with Matridge via XMPP at port 5222.
- Any connection to port 5269 attempting to use federation.

A Matridge user being able to impersonate another user or being able to impersonate an admin user and being able to run admin commands is seen as the most concerning threat.

3.3 Data Flow

Component communication

After Matridge starts, the gateway is established in the following way:

1. Matridge connects to XMPP server: `slidge/slidge/main.py:140-150` initiates connection to port 5347
2. Authentication via component secret: `slidge/slidge/core/gateway.py` validates component secret

3. User registration flow:
    - User fills XMPP registration form: `slidge/slidge/core/dispatcher/registration.py:82` handles `/register`
    - Matridge validates Matrix credentials: `matridge/gateway.py:75-94` (`validate` method):
      – Creates `AuthenticationClient` via `matridge/matrix.py` - Matrix client implementation
      – Calls `await client.login(password)` at `matridge/gateway.py:88-91`
      – Returns `Credentials` via `client.get_credentials(resp)` at `matridge/gateway.py:94`
    - Credentials stored in SQLite: `slidge/slidge/db/models.py` defines the `GatewayUser` model.

4. Message bridging:
    - XMPP → Matrix: `slidge/slidge/core/session.py:100-200` handles incoming XMPP messages
    - Matrix → XMPP: `matridge/session.py:110-270` sends messages to XMPP clients

Command authorization

Any command a user wants to execute is checked:

1. Message received: `slidge/slidge/command/chat_command.py:152` (`_handle_message`)
   Parses message body using string splitting
2. Command parsed: `slidge/slidge/command/chat_command.py:165-170`
   `first_word = body.split()[0]` extracts command name
   `*rest = body.split()[1:]` extracts arguments
3. Authorization check: `slidge/slidge/command/base.py:452` (`raise_if_not_authorized`)
   Checks `CommandAccess` at `slidge/slidge/command/base.py:224-235`
   Verifies `is_admin()` at `slidge/slidge/command/base.py:560-561`
4. Command execution: `slidge/slidge/command/base.py:427-440` (`run` method)
   Calls command-specific `run()` implementation
5. Response delivered via XMPP message: `slidge/slidge/command/chat_command.py:180-210`

Command handlers

We identified the following command handlers which can potentially process attacker-controlled data:

Command	Controlled data	Slidge source	Matridge source
help [cmd]	command name string	`chat_command.py:168-169` – detects "help", `:370-406` – `_handle_help()` and `_help()`	–
find terms	`search_terms` string	`command/user.py:35-68` – `Search.run()`	`contact.py:60-74` – `Roster.find()`

preferences	preference values	`command/user.py:285-337` – `Preferences.run()` and `finish()`	`gateway.py:49-54`
unregister	`confirmation` string	`command/user.py:340-359` – `Unregister.run()`	`gateway.py:96-98` – `Gateway.unregister()`
create-group name	`group_name` string, contact JIDs	`command/user.py:225-282` – `CreateGroup.run()` and `finish()`	–
leave-group group	group JID	`command/user.py:362-390` – `LeaveGroup.run()` and `finish()`	`session.py:307-315` – `on_leave_group()`
sync-contacts	confirmation	`command/user.py:71-142` – `SyncContacts.run()` and `sync()`	–
verify [args]	device IDs, trust state	–	`command.py:81-192` – `ManageTrust` class
change-password	`current_password`, `new_password`	–	`command.py:193-279` – `ChangePassword` class
delete-devices	device IDs (form)	–	`command.py:279-394` – `DeleteDevices` class
import-keys	JSON key data (form)	–	`command.py:520-655` – `ImportRoomKeys` class
join-room room	room alias or room ID	–	`command.py:394-464` – `JoinRoom` class

4 Findings

We have identified the following issues:

4.1 CLN-001 — Cleartext storage of sensitive Information

Vulnerability ID: CLN-001

Vulnerability type: CWE-316: Cleartext Storage of Sensitive Information in Memory

Threat level: Moderate

Description:

When a user registers with the gateway, the Matrix username and password are stored in plaintext in the Matridge python process memory. Cleartext credentials are also stored in a SQLite DB.

Technical description:

The `validate()` method at `matridge/gateway.py:75-95` is called on registering a Matrix user to the gateway using the login credentials for a Matrix service. The clear text of the credentials is visible in the variables `registration_form["user"]`,`registration_form["password"]`

The same credentials are saved to a SQLite DB to disk in clear text to keep the user registered for subsequent slidge sessions and connections to Matrix.

The following snippet demonstrates the issue:

```
strings <slidge_homedir>/slidge.sqlite | grep token
```

The output reveals a string containing the `access_token`, `user_id` and `username` of Matrix credentials of users registered to the gateway.

Impact:

- Plaintext Matrix credentials are available in the SQLite database at `<slidge_homedir>/slidge.sqlite`
- An attacker with file system access to `<slidge_homedir>/` can extract all credentials
- Full Matrix account takeover could be possible after obtaining access credentials

Recommendation:

- Encrypt credentials before storing them to disk using cryptographic primitives such as Fernet.

4.2 CLN-002 — Uncaught exception on registration

Vulnerability ID: CLN-002

Vulnerability type: CWE-248: Uncaught Exception

Threat level: Moderate

Description:

An authenticated XMPP user can crash Matridge when registering to the gateway.

Technical description:

When XMPP clients want to register to Matridge to be able to connect to a Matrix chat service, they need to specify a Matrix URL, username and password. When the domain of the URL can't be resolved, an unhandled exception forces Slidge to exit:

1. At `matridge/matridge/matrix.py:94` the method `discovery_info()` of the `nio.AsyncClient` class is called during registration.
2. An `aiohttp.client_exceptions.ClientConnectorDNSError` exception is thrown but not handled. It bubbles up to `AuthenticatonClient.fix_homeserver()` in `matridge/matridge/matrix.py:94`, is not caught, and leads to termination of the Matridge python process.

Reproducing the issue can be achieved with an XMPP client (such as gajim) by specifying an unresolvable domain in the URL. Executing the helper script `matrix_registration.py` also triggers the issue for an XMPP user (`test@target-example.com`) by specifying a domain which does not exist as first parameter:

```
matrix_registration.py nosuchdomainxyz nobody none
```

Impact:

- Denial of service: A single malformed registration attempt by a remote attacker crashes the gateway.
- Restart required: administrator must restart the Matridge service for it to recover.

- Availability impact: All users lose access until restart.

Recommendation:

- Catch the exception with a `try except` statement wrapped around `self.discovery_info()` and handle it gracefully.

4.3 CLN-005 — Possible outdated dependencies

Vulnerability ID: CLN-005

Vulnerability type: CWE-1395: Dependency on Vulnerable Third-Party Component

Threat level: Moderate

Description:

pip-audit reveals outdated dependencies.

Technical description:

A `pip-audit` dependency scan found 43 known vulnerabilities in 10 packages. A full list is available in `pip_audit.md`

Impact:

Dependent on the targeted vulnerability, resource exhaustion or denial-of-service conditions may be triggered.

Recommendation:

- Integrate `pip-audit` into CI and keep packages up to date so that security patches are installed as soon as vulnerabilities in dependencies are reported and fixed.

5 Non-Findings

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

5.1 NF-003 — Potential use of exec() may lead to arbitrary code execution

Potential CWE-95: Improper Neutralization of Directives in Dynamically Evaluated Code ('Eval Injection')

The call to `exec()` in `slidge/slidge/command/admin.py:215` potentially allows arbitrary code execution with the privileges of the Matridge python process. While it is guarded for admin-use only and is not used in production, it still remains an issue as long as it is part of the deployed application as it might become reachable through other means.

If a user manages to gain the privileges of an XMPP admin user, it may lead to remote code execution on the server with the privileges of the Matridge python process in case the functionality was left enabled unintentionally.

We recommend removing the `Exec` class in `slidge/slidge/command/admin.py:196`.

5.2 NF-004 — Potential arbitrary file write

Dependent on the Matridge configuration, file uploads are saved to disk at `slidge/slidge/core/mixins/attachment.py:139-158`. It is not clear whether this code:

```
destination = destination_dir / name
        method = config.NO_UPLOAD_METHOD
        if method == "copy":
            shutil.copy2(file_path, destination)
```

may lead to arbitrary file write with the privileges of the Matridge python process on the server, as `destination` might not be properly sanitized.

5.3 NF-006 — python-olm is deprecated

The olm library which is used by the dependency python-olm is deprecated and shouldn't be used. Also, it has unfixed vulnerabilities. Hence, as development on matrix-nio seems to have halted, we recommend switching to Vodozemac.

5.4 NF-007 — Bandit scan

We used Bandit to automatically scan Slidge and Matridge. It found a total of 41 issues:

- LOW: 32
- MEDIUM: 1

- HIGH: 8

The report is available as `bandit.html` in the project repo.

6 Future Work

- **SQL Assessment**
  While the application use SQLite, and we found cleartext storage/usage of user credentials, we did not perform exhaustive assessment of the database due to time constraints.
- **Stateful command handling assessment**
  Due to limited time for code review, we recommend performing a full stateful chat command assessment.
- **Retest of findings**
  When mitigations for the vulnerabilities described in this report have been deployed, perform a repeat test to ensure that they are effective and have not introduced other security problems.
- **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.

7 Conclusion

We discovered 3 Moderate-severity issues during this penetration test.

While we observed that the `mfrom=JID` attribute in chat commands (stanzas) sent to XMPP by clients are forgeable, the actual JID of the logged-in user is set before it is passed to Matridge. Hence, authentication checks for commands use the actual JID, and we did not find a way to impersonate it.

For federation attacks we emulated a server making s2s connections from another domain to the target server. While we were able to set a custom node and trigger message/chat command handling with a forged `mfrom` attribute, we were not able to run commands as another user or with higher privileges such as admin. We observed correct bailouts on crafted messages.

We recommend fixing all of the issues found and then performing a retest in order to ensure that mitigations are effective and that no new vulnerabilities have been introduced.

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

Robert Gawlik *(pentester)*	Robert is a senior security researcher at Radically Open Security.
Melanie Rieback *(approver)*	Melanie Rieback is a former Asst. Prof. of Computer Science from the VU, who is also the co-founder/CEO of Radically Open Security.

Front page image by dougwoods (https://www.flickr.com/photos/deerwooduk/682390157/), "Cat on laptop", Image styling by Patricia Piolon, https://creativecommons.org/licenses/by-sa/2.0/legalcode.