RADICALLY OPEN SECURITY

Penetration Test Report

Yaal Coop

V 1.0
Amsterdam, March 13th, 2025
Public

Document Properties

Client	Yaal Coop
Title	Penetration Test Report
Target	• Canaille, a really easy-to-use OpenID Connect server, based on an LDAP directory
Version	1.0
Pentester	Andrea Jegher
Authors	Andrea Jegher, Marcus Bointon
Reviewed by	Marcus Bointon
Approved by	Melanie Rieback

Version control

Version	Date	Author	Description
0.1	February 14th, 2025	Andrea Jegher	Initial draft
0.2	February 19th, 2025	Marcus Bointon	Review
1.0	March 13th, 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**	**4**
1.1	Introduction	4
1.2	Scope of work	4
1.3	Project objectives	4
1.4	Timeline	4
1.5	Results In A Nutshell	4
1.6	Summary of Findings	5
1.6.1	Findings by Threat Level	6
1.6.2	Findings by Type	6
1.7	Summary of Recommendations	7
2	**Methodology**	**8**
2.1	Planning	8
2.2	Risk Classification	8
3	**Reconnaissance and Fingerprinting**	**10**
4	**Findings**	**11**
4.1	CLN-001 — Password reset link depends on HTTP request Host header	11
4.2	CLN-002 — Weak cryptographic primitive	13
4.3	CLN-004 — Insecure password reset	15
4.4	CLN-007 — Lack of content security policy	17
4.5	CLN-003 — Username enumeration	18
4.6	CLN-008 — Server-side request forgery in logo() function	20
5	**Non-Findings**	**22**
5.1	NF-009 — URL validation with regexp	22
5.2	NF-010 — Password backend storage	22
6	**Future Work**	**24**
7	**Conclusion**	**25**
Appendix 1	**Testing team**	**26**

1 Executive Summary

1.1 Introduction

Between February 10, 2025 and February 18, 2025, Radically Open Security B.V. carried out a penetration test for Yaal Coop.

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:

- Canaille, a really easy-to-use OpenID Connect server, based on an LDAP directory

The scoped services are broken down as follows:

- Canaille, an OpenID Connect server really easy to use, based on an LDAP directory: 6 days
- PM/Review: 1.5 days
- **Total effort: 7.5 days**

1.3 Project objectives

ROS will perform a penetration test of Canaille with Yaal Coop in order to assess the security of its implementation. To do so ROS will access its public source code repository and guide Yaal Coop in attempting to find vulnerabilities, exploiting any such found to try and gain further access and elevated privileges.

1.4 Timeline

The security audit took place between February 10, 2025 and February 18, 2025.

1.5 Results In A Nutshell

During this crystal-box penetration test we found 1 High, 3 Moderate and 2 Low-severity issues.

The password reset link generation relies on the HTTP request `Host` header when `_external` is set to true in `Flask.url_for`. Since users can manipulate the `Host` header, this could lead to account takeovers by directing users to malicious password reset URLs CLN-001 (page 11).

The password reset mechanism follows insecure practices, potentially allowing unauthorized password changes or account takeovers CLN-004 (page 15). Additionally, username enumeration is possible due to different error messages returned when an account exists or not, which can help attackers discover valid usernames CLN-003 (page 18).

A weak cryptographic primitive is used for message signing; SHA-256 with a prepended secret key is not a secure method for authentication or integrity verification CLN-002 (page 13). This could allow attackers to forge or manipulate messages under certain conditions.

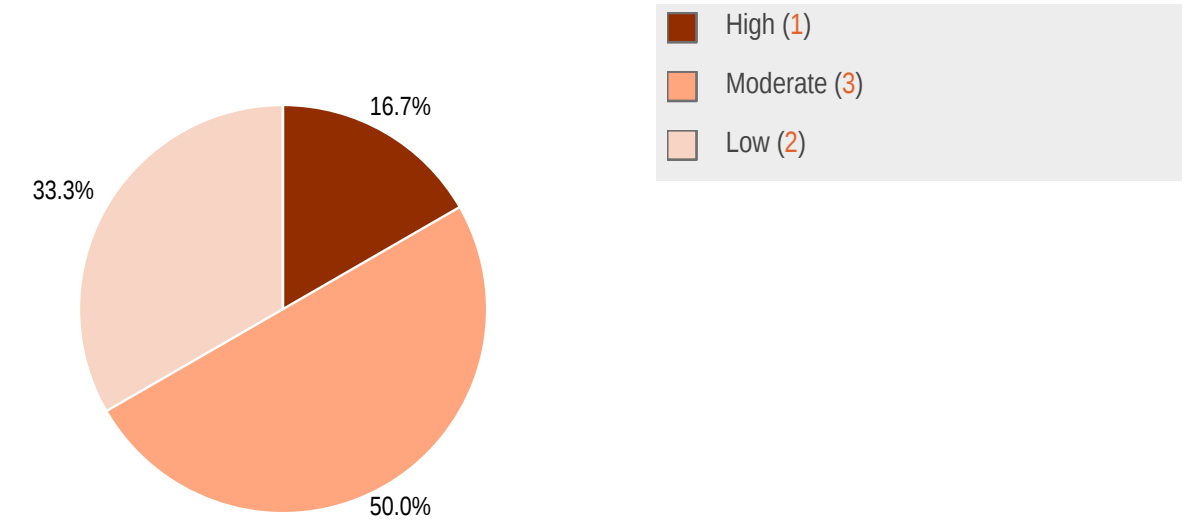
The application is also vulnerable to server-side request forgery (SSRF) in the `logo()` function CLN-008 (page 20). If `SERVER_NAME` is not set, the function fetches the logo from a URL based on the HTTP request, which could allow attackers to make unauthorized requests to internal services.

Furthermore, the lack of a Content Security Policy (CSP) in the Flask web application increases the risk of cross-site scripting (XSS) and other client-side attacks by allowing the execution of potentially harmful scripts from untrusted sources CLN-007 (page 17).

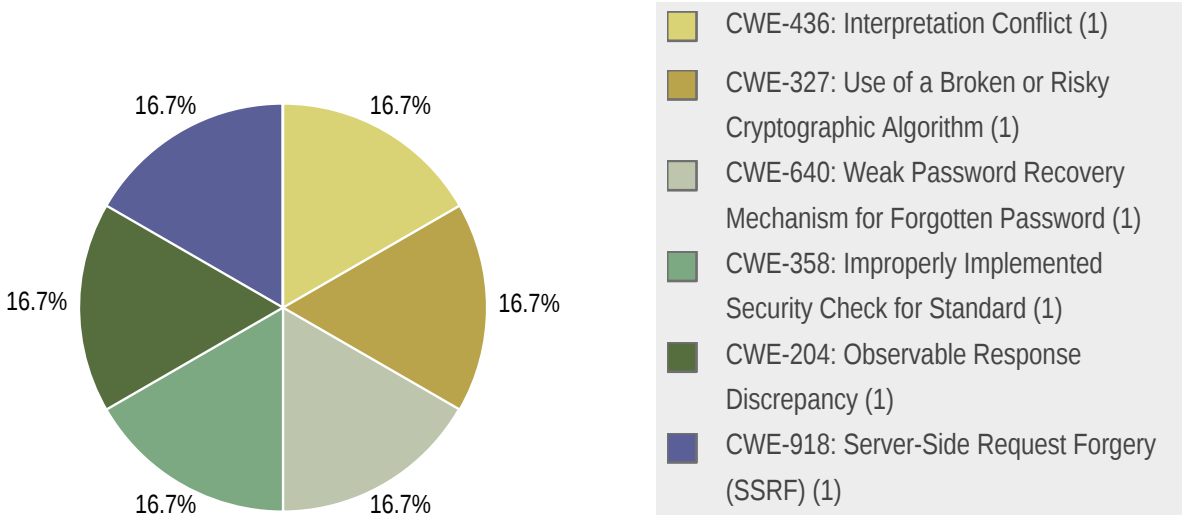
1.6 Summary of Findings

ID	Type	Description	Threat level
CLN-001	CWE-436: Interpretation Conflict	When the _external flag is true the function Flask.url_for will use the request Host header, controlled by the user, to create the final URL. In this application it might lead to an account takeover.	High
CLN-002	CWE-327: Use of a Broken or Risky Cryptographic Algorithm	The application signs messages using SHA-256, prepending a secret key to them.	Moderate
CLN-004	CWE-640: Weak Password Recovery Mechanism for Forgotten Password	The password reset mechanism design follows insecure practices.	Moderate
CLN-007	CWE-358: Improperly Implemented Security Check for Standard	The Flask web application does not implement a Content Security Policy	Moderate
CLN-003	CWE-204: Observable Response Discrepancy	The application provides different error messages depending on whether account exists or not.	Low
CLN-008	CWE-918: Server-Side Request Forgery (SSRF)	The function providing the application logo fetches it from a URL depending on the HTTP request URL if SERVER_NAME is not set in the configuration.	Low

1.6.1 Findings by Threat Level



1.6.2 Findings by Type



1.7 Summary of Recommendations

ID	Type	Recommendation
CLN-001	CWE-436: Interpretation Conflict	• Never use the `_external=True` flag, and require the system admins to set it in a configuration variable. • Alternatively, change the reset feature so that 1) the application does not send a link but the reset token only 2) Add a page where users can submit reset tokens, with their usernames, that handles the reset.
CLN-002	CWE-327: Use of a Broken or Risky Cryptographic Algorithm	• Use `HMAC` with SHA-256 to sign the message.
CLN-004	CWE-640: Weak Password Recovery Mechanism for Forgotten Password	• Redesign the reset flow following the OWASP implementation guidelines.
CLN-007	CWE-358: Improperly Implemented Security Check for Standard	• Implement a restrictive CSP header to limit allowed sources for scripts, styles, and other resources.
CLN-003	CWE-204: Observable Response Discrepancy	• Adjust responses so that they are the same regardless of whether a user exists or not.
CLN-008	CWE-918: Server-Side Request Forgery (SSRF)	• Read the default logo from the file system if the `SERVER_NAME` is not set instead of using the `request.url_root`, or disable that feature when it is not set.

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.

- Gitleaks – https://github.com/gitleaks/gitleaks
- CodeQL – https://codeql.github.com
- OpenGrep – https://github.com/opengrep/opengrep
- Bandit – https://bandit.readthedocs.io

4 Findings

We have identified the following issues:

4.1 CLN-001 — Password reset link depends on HTTP request Host header

Vulnerability ID: CLN-001

Vulnerability type: CWE-436: Interpretation Conflict

Threat level: High

Description:

When the `_external` flag is true the function `Flask.url_for` will use the request `Host` header, controlled by the user, to create the final URL. In this application it might lead to an account takeover.

Technical description:

For this issue we used the demo application at `canaille/demo/docker-compose.yml` hosted on a local machine.

The password reset feature at endpoint `/reset`, handled by code from line 86 of `canaille/core/endpoints/auth.py` accepts an input form named `ForgottenPasswordForm` that contains a field called `Login`.

A legitimate user will fill the `Login` field with their username and the application will search that username in the back end, find the email address associated with that username and send a reset link. This link includes the username and a token. Note, that the only information required to perform a password reset is the username, and this value is not meant to be private.

To send the reset password email the application uses function `send_password_reset_mail` on line 40 of file `canaille/core/mails.py`. On line 44, the application creates the reset URL with the following code. Note that the `url_for` has its `_external` parameter set to `True`.

```
reset_url = url_for(
    "core.auth.reset",
    user=user,
    hash=build_hash(
        user.identifier,
        mail,
        user.password if user.has_password() else "",
    ),
    _external=True,
)
```

When `_external` is set to `True`, Flask will use the HTTP request `Host` header for the domain of the URL. So for example if a user modifies the `Host` header in the request sent to the reset URL, the user will contain the domain set

in the request's `Host` header. For example, in the following curl command we set the `Host` header to `attacker.com` (the cookie and CSRF token values are required but tied to a non-authenticated user session, so anyone can send this request for any username):

```
curl --path-as-is -i -s -k -X $'POST' \
    -H $'Host: attacker.com' \
    -H $'Content-Type: multipart/form-data; boundary=----WebKitFormBoundary4h8QAnJFRxIJFx9h' \
    -b $'canaille=COOKIE' \
    --data-binary $'------WebKitFormBoundary4h8QAnJFRxIJFx9h\x0d\x0aContent-Disposition: form-data;
 name=\"csrf_token\"\x0d\x0a\x0d\x0aCSRF_TOKEN\x0d\x0a------WebKitFormBoundary4h8QAnJFRxIJFx9h
\x0d\x0aContent-Disposition: form-data; name=\"login\"\x0d\x0a\x0d\x0aadmin\x0d\x0a------
WebKitFormBoundary4h8QAnJFRxIJFx9h--\x0d\x0a' \
    $'http://canaille.domain/reset'
```

That will generate an HTTP request similar to this:

```
POST /reset HTTP/1.1
Host: attacker.com
Content-Length: 329
Content-Type: multipart/form-data; boundary=----WebKitFormBoundary4h8QAnJFRxIJFx9h
Cookie: canaille=COOKIE

------WebKitFormBoundary4h8QAnJFRxIJFx9h
Content-Disposition: form-data; name="csrf_token"

CSRF_TOKEN
------WebKitFormBoundary4h8QAnJFRxIJFx9h
Content-Disposition: form-data; name="login"

admin
------WebKitFormBoundary4h8QAnJFRxIJFx9h--
```

As a consequence of the reset password request, the application mail server will send the following email. Note, that the URL host is `attacker.com`, the same as the one set in our curl request, and the URL contains the password reset token too.

```
# Password initialization

In order to finalize your account configuration at Canaille, we need to setup your password. Please
 click on the link below and follow the instructions.

Initialize password: http://attacker.com/reset/admin/
aacf6d8ef4ed72421885ec41615259909d6da775f5e06c26e6b58f89900d1edd
Canaille: http://attacker.com/
```

Impact:

If a user receiving this email clicks on the link, they will visit the `attacker.com` website instead of the original Canaille instance, leaking the reset token to a third party. This third party can then use the reset token to reset the user password and take over their account.

Note that there are three mitigations:

1. The attacker must know the username of a user
2. A user must click on the reset link in the email, which they might not
3. If there is a web server in front of the application, like Nginx, it might reject the request since the `Host` header does not match any of the domains it is configured to serve

Recommendation:

- Never use the `_external=True` flag, and require the system admins to set it in a configuration variable. This is a very drastic change in the design of the application but any time the application uses this flag it will use the request `Host` header that an attacker could exploit.
- Alternatively, change the reset feature so that 1) the application does not send a link but the reset token only 2) Add a page where users can submit reset tokens, with their usernames, that handles the reset. In this way even if an attacker submits a reset for another user it is the final user that decides the domain to visit.

4.2 CLN-002 — Weak cryptographic primitive

Vulnerability ID: CLN-002

Vulnerability type: CWE-327: Use of a Broken or Risky Cryptographic Algorithm

Threat level: Moderate

Description:

The application signs messages using SHA-256, prepending a secret key to them.

Technical description:

Function `build_hash` at file `canaille/app/__init__.py` line 23 creates hash of input arguments from the app secret key using SHA-256 algorithm.

```
def build_hash(*args):
    return hashlib.sha256(
        current_app.config["SECRET_KEY"].encode("utf-8")
        + obj_to_b64(str(args)).encode("utf-8")
    ).hexdigest()
```

Endpoints like `/register` (in `canaille/core/endpoints/account.py` on line 235) uses this function to verify that an argument (data) was not altered by the user after receiving it from the server.

```
@bp.route("/register", methods=["GET", "POST"])
@bp.route("/register/<data>/<hash>", methods=["GET", "POST"])
def registration(data=None, hash=None):
...
        if hash != payload.build_hash():
            flash(
                _("The registration link that brought you here was invalid."),
                "error",
            )
            return redirect(url_for("core.account.index"))
```

The `data` parameter in this function could for example come from function `user_invitation` in the same file at line 199, and contains a new user initial fields that the server generates and signs using the `build_hash` function.

Impact:

The function directly concatenates the `SECRET_KEY` with the hashed data before passing it to sha256, a kind of naïve MAC calculation. This makes the hash predictable and could lead to security flaws such as hash extension attacks.

An extension attack means that an attacker who knows `hash(secret + message)` and message can compute `hash(secret + message + appended_message)` without knowing `SECRET_KEY`.

While this attack is feasible, we did not find a message to append that could have an impact on the application since the `message` (`data` in the route) must be proper JSON.

Recommendation:

Use `HMAC` with SHA-256 to sign the message. For example, with the following code:

```
            import hmac
                import hashlib

                def build_hash(*args):
                key = current_app.config["SECRET_KEY"].encode("utf-8")
                message = obj_to_b64(str(args)).encode("utf-8")
                return hmac.new(key, message, hashlib.sha256).hexdigest()
```

4.3 CLN-004 — Insecure password reset

Vulnerability ID: CLN-004

Vulnerability type: CWE-640: Weak Password Recovery Mechanism for Forgotten Password

Threat level: Moderate

Description:

The password reset mechanism design follows insecure practices.

Technical description:

The password reset mechanism is as follows.

1. A user visits the "/reset" endpoint and submits their email address
2. The application sends an email to the user, if they are registered, through function `send_password_reset_mail` at file `canaille/core/mails.py` line 40. This function creates a reset URL and sends the email. The following line of code generates the reset URL. Note that, `build_hash` uses an application secret and is not a simple hash.

```
reset_url = url_for(
    "core.auth.reset",
    user=user,
    hash=build_hash(
        user.identifier,
        mail,
        user.password if user.has_password() else "",
    ),
    _external=True,
)
```

3. When the user clicks the link in the email they will visit endpoint `/reset/<user:user>/<hash>`. This endpoint is handled in file `canaille/core/endpoints/auth.py` on line 244, and checks that the hash in the request URL matches with the one calculated in the same way, in order to validate it. We report the relevant code, where `hash` is the parameter in the request URL and the `hashes` is the list of hashes, in case of multiple emails, the server uses to validate it.

```
hashes = {
        build_hash(
            user.identifier,
            email,
```

```
            user.password if user.has_password() else "",
        )
        for email in user.emails
    }
    if not user or hash not in hashes:
        flash(
            _("The password reset link that brought you here was invalid."),
            "error",
        )
        return redirect(url_for("core.account.index"))
```

On this mechanism we make the following considerations:

1. The reset token does not expire
2. The reset token value changes only after a password change or an application secret key (since the `build_hash` function uses the application secret)
3. The reset is not random but tied to user content
4. The application does not invalidate any existing user sessions on a successful reset
5. There is a time window when a user is invited when the password is not set, when the reset token might be predictable

Impact:

An attacker might take advantage of the insecure design and reset the password for a different user.

Recommendation:

Redesign the reset flow following the OWASP implementation guidelines. That are in short:

1. Ensure that generated tokens or codes are:
    1. Randomly generated using a cryptographically safe algorithm
    2. Sufficiently long to protect against brute-force attacks
    3. Stored securely, e.g. in a database
    4. Single use and expire after an appropriate period
2. Invalidate the sessions automatically after a successful reset

4.4 CLN-007 — Lack of content security policy

Vulnerability ID: CLN-007

Vulnerability type: CWE-358: Improperly Implemented Security Check for Standard

Threat level: Moderate

Description:

The Flask web application does not implement a Content Security Policy

Technical description:

Content Security Policy (CSP) is an HTTP header that tells the browser the sources from which a web application can load scripts, styles, and other resources. One of its primary goals is in mitigating cross-site scripting (XSS) attacks.

To reproduce this issue run the local demo container `demo/docker-compose.yml` and perform a `curl` request like the following. From the response it is possible to see that HTTP security headers, including CSP, are not present in the response.

```
curl -I https://localhost:5000
```

Additionally, perform the same request to the official live demo, where it is possible to see that the only HTTP security header missing is the CSP.

```
curl -I https://demo.canaille.yaal.coop/login
```

As other HTTP security headers are easy to add from a web server separate from the application, CSP may need to be set from the application itself. For example, to generate a random nonce for each script.

Impact:

The application does not mitigate common web application attacks. On its own it is not a threat to the application, but increases the risk of successful exploitation should other vulnerabilities arise.

Recommendation:

Implement a restrictive CSP header to limit allowed sources for scripts, styles, and other resources. For example, using Talisman that integrates with Flask.

Other resources to better understand CSP are:

1. OWASP Cheat Sheet
2. Mozilla CSP Docs
3. Google CSP Validator

4.5 CLN-003 — Username enumeration

Vulnerability ID: CLN-003

Vulnerability type: CWE-204: Observable Response Discrepancy

Threat level: Low

Description:

The application provides different error messages depending on whether account exists or not.

Technical description:

User enumeration vulnerabilities occur when an attacker can distinguish between valid and invalid usernames based on system responses. We found several endpoints where the application responds differently depending on whether a username or email exists.

1. "/register"

This function, found in file `canaille/core/endpoints/account.py` on line 235, handles user registration. One of the steps at line 339 is to validate the input form fields. Two validators, `unique_email` and `unique_user_name`, defined in `canaille/core/validators.py` lines 11 and 20, validates whether the submitted username and email are not already in the system.

```
def unique_user_name(form, field):
    if Backend.instance.get(models.User, user_name=field.data) and (
        not getattr(form, "user", None) or form.user.user_name != field.data
    ):
        raise wtforms.ValidationError(
            _("The user name '{user_name}' already exists").format(user_name=field.data)
        )


def unique_email(form, field):
    if Backend.instance.get(models.User, emails=field.data) and (
        not getattr(form, "user", None) or field.data not in form.user.emails
    ):
        raise wtforms.ValidationError(
            _("The email '{email}' is already used").format(email=field.data)
```

```
        )
```

When any of these validators fail they raise a validation error that is included in the HTTP response, containing a message stating that the email or the username already exists.

2. Endpoints with `<user:user>` variable in path that do not require OAuth permission

```
canaille/core/endpoints/account.py
  867,21: @bp.route("/profile/<user:user>/<field>")
  889,19: @bp.route("/reset/<user:user>", methods=["GET", "POST"])

canaille/core/endpoints/auth.py
  159,24: @bp.route("/firstlogin/<user:user>", methods=("GET", "POST"))
  243,19: @bp.route("/reset/<user:user>/<hash>", methods=["GET", "POST"])
```

The application will respond to unauthenticated requests with a `404 not found` response when the user does not exist, a `403` when the user exists and the request contains a valid CSRF token and cookie, and a `400` if the CSRF token and cookie are not present or invalid.

To reproduce this issue you could use the following `curl` command, first using `admin` and then `adminx` in the username, observing two different responses (one a 400 and the second a 404).

```
curl --path-as-is -i -s -k -X $'POST' \
    -H $'Content-Type: application/x-www-form-urlencoded' \
    -H $'Content-Length: 12' \
    -b $'canaille=x' \
    --data-binary $'csrf_token=x' \
    $'http://canaille.domain/firstlogin/admin'
```

Impact:

An attacker can enumerate valid usernames and email. This can be further exploited with other vulnerabilities, to conduct credential stuffing or social engineering attacks.

Recommendation:

The "/register" endpoint, must respond with the same message regardless of whether the user already exists, for example `more information has been sent to your email`; If the user does not exist, no email will sent by the system.

For the endpoints with `<user:user>` follow these examples:

1. Route `/profile/<user:user>/<field>` should return a default image for any user that has not it set or that does not exist.
2. Routes `/reset/<user:user>` and `/reset/<user:user>/<hash>` should always return an unauthorized message if the user does not exist or the request does not have the permissions to reset the password.

3. Route `/firstlogin/<user:user>` must return the same message when the user does not exist and when their password is already set.

4.6 CLN-008 — Server-side request forgery in logo() function

Vulnerability ID: CLN-008

Vulnerability type: CWE-918: Server-Side Request Forgery (SSRF)

Threat level: Low

Description:

The function providing the application logo fetches it from a URL depending on the HTTP request URL if `SERVER_NAME` is not set in the configuration.

Technical description:

Note that in order for this issue to work the `SERVER_NAME` configuration variable must be set. In the last version of the code tested, the application did not work when it was not set.

Server-Side Request Forgery (SSRF) occurs when an application lets an attacker send requests from the server. In this case, a `GET` HTTP request is sent without headers or cookies. The application uses the `logo()` function to fetch an image included in email attachments.

For example, when a user registers, the server processes their email and sends a confirmation email with a registration link. This email also includes the Canaille logo, which is retrieved from `/static/img/canaille-head.webp`.

Function `logo()` defined in file `canaille/app/mails.py` at line 14, fetches, depending on some configuration, the application logo to be sent in emails, and it is used by the following functions:

```
canaille/core/mails.py
  13,41:     logo_cid, logo_filename, logo_raw = logo()
  52,41:     logo_cid, logo_filename, logo_raw = logo()
  93,41:     logo_cid, logo_filename, logo_raw = logo()
  124,41:     logo_cid, logo_filename, logo_raw = logo()
  155,41:     logo_cid, logo_filename, logo_raw = logo()
  186,41:     logo_cid, logo_filename, logo_raw = logo()
  219,41:     logo_cid, logo_filename, logo_raw = logo()
  256,41:     logo_cid, logo_filename, logo_raw = logo()
```

In the demo application, the final URL is assigned on line 28 of `canaille/app/mail.py`, and later flows into the `urllib.request.urlopen(logo_url)` function that, depending on the URL's scheme, fetches the resource.

```
if current_app.config.get("SERVER_NAME"):
            logo_url = "{}://{}/{}".format(
```

```
                current_app.config.get("PREFERRED_URL_SCHEME"),
                get_current_domain(),
                logo_url,
            )
        else:
            logo_url = f"{request.url_root}{logo_url}"

    try:
        with urllib.request.urlopen(logo_url) as f:
            logo_raw = f.read()
```

Note, that the `logo_url` value depends on `request.url_root`, which is controlled by the user performing this request. For example, it could be possible to create an HTTP request with this host header `Host: other.internal.server/resource.txt?x=` to make the application fetch the `resources.txt` file instead of the logo. Then this resource will be added to the email attachments and sent to the attacker.

Impact:

Depending on the environment, an attacker might be able to extract sensitive information.

Several restrictions reduce the risk of this issue. For example, Flask always prepends at least `http://` to `request.url_root`, preventing the use of arbitrary URL schemes like `file://` to access the file system.

As a result, the SSRF target must be an HTTP server, such as the AWS metadata endpoint if the application is hosted on an EC2 instance. However, since no headers can be set, only very old EC2 instances using the first version of the metadata endpoint would be vulnerable.

The only viable target would be another internal server exposing sensitive information without authentication. Even in this case, the attacker would need to know the exact path to access any sensitive data.

Recommendation:

- Read the default logo from the file system if the `SERVER_NAME` is not set instead of using the `request.url_root`, or disable that feature when it is not set.

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-009 — URL validation with regexp

Using regular expressions for URL validation can introduce security risks such as overly permissive patterns, bypassing, denial of service (DoS) attacks, and inconsistent validation.

For example, the regexp from function `validate_uri` in file `canaille/app/__init__.py` at line 54, validates URLs submitted by a user.

```
def validate_uri(value):
    regex = re.compile(
        r"^(?:[A-Z0-9\\.-]+)s?://"  # scheme + ://
        r"(?:(?:[A-Z0-9](?:[A-Z0-9-]{0,61}[A-Z0-9])?\.)+(?:[A-Z]{2,6}\.?|[A-Z0-9-]{2,}\.?)|"  #
 domain...
        r"[A-Z0-9\\.-]+|"  # hostname...
        r"\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3})"  # ...or ip
        r"(?::\d+)?"  # optional port
        r"(?:/?|[/?]\S+)$",
        re.IGNORECASE,
    )
    return re.match(regex, value) is not None
```

This pattern does not properly validate the scheme (e.g., HTTP, HTTPS), allowing potentially harmful URLs like `data://` or `file://`. Since the application does not use this information, other than for displaying in the UI, an attacker cannot exploit this lack of validation, especially because the scheme must contain `://`, preventing a URL like `javascript:alert()` from being valid.

A safer approach is to use the Python `urllib.parse` module, which provides robust URL parsing and validation, as in the following example snippet. This could also allow for more restrictive filtering, like removing query parameters:

```
from urllib.parse import urlparse

def is_valid_url(url):
    parsed = urlparse(url)
    return parsed.scheme in ['http', 'https'] and bool(parsed.netloc)
```

5.2 NF-010 — Password backend storage

The application uses three different back ends for data storage: SQL, LDAP, and in-memory. Each backend has its own password storage method.

1. SQL Backend

The SQL backend uses the `sqlite` hashing mechanism combined with PBKDF2 (Password-Based Key Derivation Function 2). PBKDF2 is a strong hashing algorithm that enhances security by making brute-force attacks more difficult through its iterative nature.

2. In-Memory Backend

Currently, the in-memory back end stores passwords in plaintext. This is a significant security risk as anyone with access to the running application server can retrieve these passwords. Consider transitioning to PBKDF2, as used in the SQL back end, to hash passwords before storing them in memory.

3. LDAP Backend

In the current setup, the demo users' passwords are stored in plaintext. However, for new users, the password is hashed using salted SHA-1, showing that the LDAP server will hash the password when asked to save a new user.

Even if the demo apps may represent an example for the final user and are not indented for production, it is important to document this behavior of the system and if possible implement policies or mitigation to ensure that the system only store users' password hash using secure algorithms like bcrypt, PBKDF2, or Argon2.

6 Future Work

- **Production-ready setup**
  The project does not include a production-ready setup. It would be beneficial to add one, at least for one backend (e.g., SQL), to provide users with an example that follows security best practices.
  This way, a future penetration test could also review the production architecture, offering additional suggestions for deploying the application securely.
- **Retest of findings**
  When mitigations for the vulnerabilities described in this report have been deployed, a repeat test should be performed 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 1 High, 3 Moderate and 2 Low-severity issues during this penetration test.

The most pressing issue pertains to the potential for account takeovers through manipulated password reset links, which rely on insecure practices within `Flask.url_for`. This flaw is exacerbated by weak cryptographic measures (SHA-256 with a prepended key) and username enumeration vulnerabilities, both heightening the risk of unauthorized access.

The remaining concerns including the method of fetching application logos and varying error message handling, which might, under certain configurations, lead to sensitive information disclosure. While not immediately catastrophic, these vulnerabilities still represent potential exposure points that could be leveraged in combination with more severe flaws.

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 – this penetration test is just a one-time snapshot. Security posture must be continuously evaluated and improved. 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

Andrea Jegher *(pentester)*	Andrea is a security engineer with experience in offensive security and secure development. He started his career focusing on Web Application as a developer and as a penetration tester. Later he studied other fields of security such as cloud, networks and desktop applications.
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.