RADICALLY OPEN SECURITY

Karrot Penetration Test

Karrot

V 1.0
Amsterdam, April 1st, 2025
Public

Document Properties

Client	Karrot
Title	Karrot Penetration Test
Targets	• https://codeberg.org/karrot/karrot-backend • https://codeberg.org/karrot/karrot-frontend • https://dev.karrot.world
Version	1.0
Pentester	Martin Cuddy
Authors	Martin Cuddy, Marcus Bointon
Reviewed by	Marcus Bointon
Approved by	Melanie Rieback

Version control

Version	Date	Author	Description
0.1	March 13th, 2025	Martin Cuddy	Initial draft
0.2	March 31st, 2025	Marcus Bointon	Review
1.0	April 1st, 2025	Martin Cuddy	Final

Contact

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

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

Table of Contents

1 Executive Summary

1.1 Introduction

Between March 7, 2025 and March 13, 2025, Radically Open Security B.V. carried out a penetration test for Karrot.

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 targets:

- https://codeberg.org/karrot/karrot-backend
- https://codeberg.org/karrot/karrot-frontend
- https://dev.karrot.world

The scoped services are broken down as follows:

- Testing: 2 days
- Reporting: 1 days
- PM/Reviewing: 0.5 days
- **Total effort: 3.5 days**

1.3 Project objectives

ROS will perform a penetration test of the Karrot application with its developers in order to assess its security. To do so ROS will access the target application and guide Karrot 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 March 7, 2025 and March 13, 2025.

1.5 Results In A Nutshell

During this crystal-box penetration test we found 3 Moderate and 6 Low-severity issues.

We noted a number of related issues surrounding authentication: there is no password policy CLN-007 (page 12) and a lack of rate limiting CLN-004 (page 17), specifically no controls on login attempts CLN-003 (page 16). These leave Karrot especially vulnerable to various kinds of brute-force-based login attacks such as password guessing and credential stuffing. A lack of application rate limiting generally also leaves the user e-mail change mechanism open to abuse CLN-001 (page 15).

We found two cases where API endpoints had broken object property level authorization, whereby users could change values they should not be able to change in CLN-002 (page 11) and CLN-009 (page 20). Users can also select any username, including those on a deny list applied to display names CLN-006 (page 19).

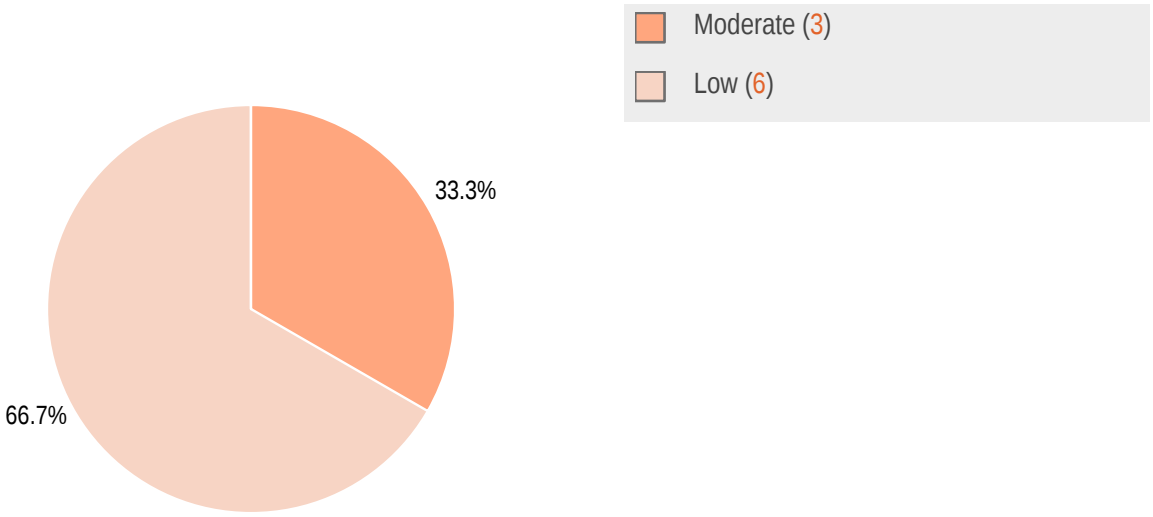
We also found that Karrot allows uploading files of any type, and files can later be accessed directly by users. These leads to a few potential abuses that we illustrated CLN-008 (page 13).

Finally we noted Karrot has numerous outdated dependencies with known security vulnerabilities CLN-005 (page 18).

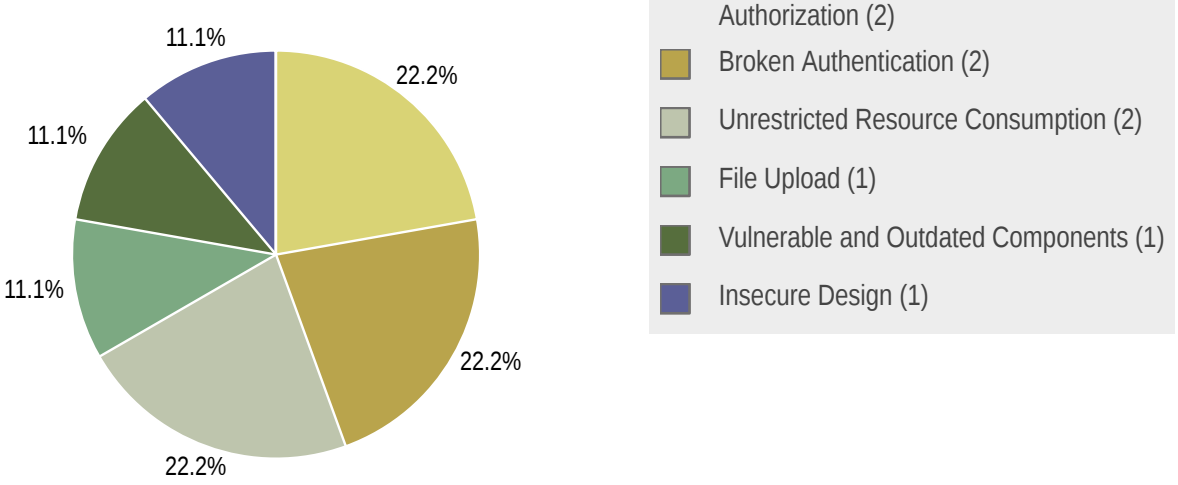
1.6 Summary of Findings

ID	Type	Description	Threat level
CLN-002	Broken Object Property Level Authorization	Users can change their username by direct PATCH request to the API.	Moderate
CLN-007	Broken Authentication	There is no password policy.	Moderate
CLN-008	File Upload	Karrot allows users to upload files of any type, with potential security consequences. We demonstrate how this can be abused in a simple phishing attack between users.	Moderate
CLN-001	Unrestricted Resource Consumption	The mechanism for changing a user's email address is open to abuse.	Low
CLN-003	Broken Authentication	The login page lacks rate limiting or other controls against brute-forcing.	Low
CLN-004	Unrestricted Resource Consumption	Karrot lacks any application-level rate limiting on requests to most endpoints. This has a variety of security consequences, some of which are outlined in separate findings.	Low
CLN-005	Vulnerable and Outdated Components	Karrot has numerous outdated python dependencies.	Low
CLN-006	Insecure Design	The hard-coded deny list of reserved names for user display names is not imposed on usernames.	Low
CLN-009	Broken Object Property Level Authorization	It's possible to change an activity's type through the API.	Low

1.6.1 Findings by Threat Level



1.6.2 Findings by Type



1.7 Summary of Recommendations

ID	Type	Recommendation
CLN-002	Broken Object Property Level Authorization	• Ensure `username` is read-only until such time as a `username` change feature is fully implemented.
CLN-007	Broken Authentication	• Implement a sensible password policy with a minimum password length. • Make use of Django's built-in password policy features.
CLN-008	File Upload	• Implement an allow-list for file upload types, only allowing file types which are considered necessary.
CLN-001	Unrestricted Resource Consumption	• Add general rate limiting on requests to API endpoints, see CLN-004 (page 17). • Implement a "cool down" period between e-mail address changes (e.g. only allow one e-mail address change per day).
CLN-003	Broken Authentication	• General rate limiting on authenticated and non-authenticated requests is recommended as in CLN-004 (page 17). • Authentication endpoints should have stricter limiting on login attempts, for example by limiting the number of attempts for a given username in a window of time. • Apply these limits to both authentication endpoints `/api/auth` and `/api-auth/login`.
CLN-004	Unrestricted Resource Consumption	• Use `UserRateThrottle` to impose sensible rate limits on all API endpoints.
CLN-005	Vulnerable and Outdated Components	• Update all dependencies, `django` and `djangorestframework` in particular. • Implement automated scanning and dependency management.
CLN-006	Insecure Design	• Apply the `RESERVED_NAMES` deny list to usernames.
CLN-009	Broken Object Property Level Authorization	• Ensure the `activity_type` field is read-only.

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.

- OWASP Zed Attack Proxy – https://github.com/zaproxy/zaproxy
- safety – https://github.com/pyupio/safety
- SonarQube – https://sonarqube.org/
- Semgrep – https://semgrep.dev/

4 Findings

We have identified the following issues:

4.1 CLN-002 — Users can change username

Vulnerability ID: CLN-002

Vulnerability type: Broken Object Property Level Authorization

Threat level: Moderate

Description:

Users can change their `username` by direct PATCH request to the API.

Technical description:

Changes to Karrot user settings (display name, location, etc) are achieved via a PATCH request to `/api/auth/user/`.

```
PATCH https://dev.karrot.world/api/auth/user/ HTTP/1.1
host: dev.karrot.world

...
{
  "display_name" : "new-display-name",
  "id" : 601
}
```

We found that the `username` can also be changed if it is included in the `json` body:

```
{
  "username" : "new-username",
  "id" : 601
}
```

Users are not otherwise able to change their `username` and this is not an intended ability or implemented feature or Karrot.

If a user deletes their account, tagged references become unlinked (this also happens if a user changes their `username` using the above mechanism). A user can subsequently change their `username` to the `username` of a deleted account, and all tagged mentions will be restored, pointing to the account that has newly assumed the `username`.

Impact:

- Users can change their `username` when they should not be able to.
- Some potential for user impersonation, particularly of deleted accounts.

Recommendation:

- Ensure `username` is read-only until such time as a `username` change feature is fully implemented.

See also: https://owasp.org/API-Security/editions/2023/en/0xa3-broken-object-property-level-authorization/.

4.2 CLN-007 — No password policy

Vulnerability ID: CLN-007

Vulnerability type: Broken Authentication

Threat level: Moderate

Description:

There is no password policy.

Technical description:

Karrot lacks any password policy. There are no requirements for password length or complexity, and no mechanism to prevent common or easily guessed passwords (e.g. using one's username as a password), leaving users able to set weak passwords.

This exacerbates the lack of anti-brute-force mechanisms on Karrot's authentication endpoints reported in CLN-003 (page 16).

Impact:

- Users are not required to use a strong password: user passwords are consequently likely to be weak and easily guessed or brute-forced.

Recommendation:

- Implement a sensible password policy with a minimum password length.
- Django has excellent built in tools for password validation: requiring minimum password length, rejecting common and full numeric passwords, and ensuring a password is not too similar to other user characteristics.
- OWASP (quoting NIST) recommends a minimum password length of 8 characters.

See also: https://owasp.org/API-Security/editions/2023/en/0xa2-broken-authentication/.

4.3 CLN-008 — Unrestricted file upload

Vulnerability ID: CLN-008

Vulnerability type: File Upload

Threat level: Moderate

Description:

Karrot allows users to upload files of any type, with potential security consequences. We demonstrate how this can be abused in a simple phishing attack between users.

Technical description:

Karrot allows users to attach files to any chat message or wall post. All file types and extensions are accepted. These attachments are accessible to other users of the same conversation or group in the case of wall posts.

The original uploaded file can be directly accessed by a user with sufficient privileges via a URI such as `/api/attachments/162/original/`.

Many different potentially dangerous files can be uploaded. Filetypes that are executed client-side, such as `.html` and `.svg`, can be uploaded and linked to so that a user might be deceived into running malicious code as in the example provided below.

Server-side executable files can also be uploaded, such as `.php` files. We generally recommended preventing the upload of such files, unless they are absolutely necessary. In a standard Karrot installation, these files are not executed as Karrot does not use PHP. However, Karrot can be deployed on the same server alongside PHP applications and in the case of a misconfiguration such files might become executable.

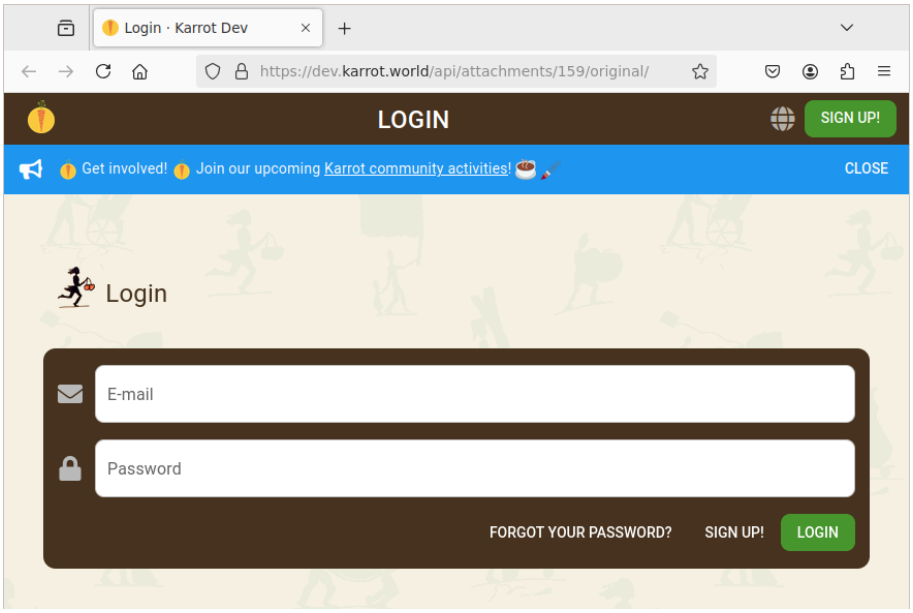
Example: uploading a malicious `.html` file to use in a phishing attack

A malicious user can clone Karrot's login page (`/#/login/`) and modify the page to collect user credentials. The resulting `.html` file can be uploaded as an attachment to a wall post, making it accessible to other members of the group.

To phish another member of the group, the malicious user can include the direct link to the `.html` attachment in an e-mail or other message to a target user:

> Hi! Check out this new file on our Karrot group: `https://karrot.world/api/attachments/162/original/`

If the target user opens the link from a browser that is logged into Karrot, they will see what appears to be a login page for Karrot:



Any credentials entered will be collected by the malicious user. After clicking "LOGIN" the target user can simply be directed to their Karrot group, making it look like their login worked successfully.

As with many phishing attacks, success relies on the target user not being vigilant, for example not noticing the URL does not point to the usual login page (although it is under the Karrot instance's domain).

This type of attack is made easier by Karrot's allowing `.html` attachments which can then be accessed and rendered by other users.

Impact:

- A lack of restrictions on file types/extensions allows dangerous or malicious files to be uploaded.

- As an example, a malicious `.html` phishing page can be uploaded to facilitate phishing attacks against other users.

Recommendation:

- Implement an allow-list for file upload types, only allowing file types which are considered necessary.

See also: https://cheatsheetseries.owasp.org/cheatsheets/File_Upload_Cheat_Sheet.html.

4.4 CLN-001 — Abuse of e-mail change mechanism

Vulnerability ID: CLN-001

Vulnerability type: Unrestricted Resource Consumption

Threat level: Low

Description:

The mechanism for changing a user's email address is open to abuse.

Technical description:

Requests by a user to change their e-mail address are accomplished via the `/api/auth/email/` endpoint:

```
PUT https://dev.karrot.world/api/auth/email/ HTTP/1.1
host: dev.karrot.world

...
{
  "new_email" : "new-email@example.com",
  "password" : "password"
}
```

Karrot sends a verification e-mail to the new e-mail address. A user can immediately request to change their e-mail again, which triggers another verification e-mail. There is no "cool down" period between requests, and a general lack of rate limiting/throttling on requests, as we mentioned in see CLN-004 (page 17).

A malicious user can abuse this mechanism to spam arbitrary e-mail addresses. E-mail change requests can be made repeatedly, alternating between two target e-mail addresses. This can be automated to send a large volume of e-mails in a short amount of time.

Impact:

- Malicious users can spam target e-mail addresses.
- Malicious users can waste Karrot's e-mail sending resources, and potentially cause reputational damage to the sending domain.

Recommendation:

- Add general rate limiting on requests to API endpoints, see CLN-004 (page 17).
- Implement a "cool down" period between e-mail address changes (e.g. only allow one e-mail address change per day).

4.5 CLN-003 — No rate limiting on authentication

Vulnerability ID: CLN-003

Vulnerability type: Broken Authentication

Threat level: Low

Description:

The login page lacks rate limiting or other controls against brute-forcing.

Technical description:

The general lack of rate limiting in Karrot CLN-004 (page 17) has significant consequences for authentication. Login attempts can be made repeatedly without any restrictions, making brute-force and credential-stuffing attacks much easier.

Impact:

- Attackers can more easily brute-force valid credentials, or attempt credential stuffing

Recommendation:

- General rate limiting on authenticated and non-authenticated requests is recommended as in CLN-004 (page 17).
- Authentication endpoints should have stricter limiting on login attempts, for example by limiting the number of attempts for a given username in a window of time. See: https://cheatsheetseries.owasp.org/cheatsheets/Authentication_Cheat_Sheet.html#protect-against-automated-attacks
- Apply these limits to both authentication endpoints `/api/auth` and `/api-auth/login`.

See also: https://owasp.org/API-Security/editions/2023/en/0xa2-broken-authentication/.

4.6 CLN-004 — General lack of rate limiting

Vulnerability ID: CLN-004

Vulnerability type: Unrestricted Resource Consumption

Threat level: Low

Description:

Karrot lacks any application-level rate limiting on requests to most endpoints. This has a variety of security consequences, some of which are outlined in separate findings.

Technical description:

Karrot uses the `djangorestframework`'s `UserRateThrottle` to rate limit user requests, but only for a handful of endpoints: applications to groups, inviting new users, opening issues, and the status endpoint. All other endpoints have no rate limiting, most notably the login page and e-mail change mechanism, which we reported in CLN-003 (page 16) and CLN-001 (page 15).

The lack of rate limiting on other endpoints and applications functions is less consequential but still of concern. An authenticated user can easily spam the various conversations (direct chats, wall posts, offers, etc) and files can be attached to these messages. Although there is a limit on 10MB for file and 6 files per message, with the lack of a rate limit, these limits will do little to hinder a malicious user who wishes to spam a group with messages or fill up server storage space with attachments.

Karrot lacks a mechanism for group editors to delete such spam messages and attachments or to deal with a malicious user quickly.

Impact:

- Specific resource-intensive operations can be abused, such as file upload and e-mail change CLN-001 (page 15).
- Brute-force attacks on authentication are made easier CLN-003 (page 16).
- Authorized users can overwhelm groups with spam.
- Denial-of-service attacks via excessive requests.

Recommendation:

- Use `UserRateThrottle` to impose sensible rate limits on all API endpoints.

See also: https://owasp.org/API-Security/editions/2023/en/0xa4-unrestricted-resource-consumption/.

4.7 CLN-005 — Outdated dependencies

Vulnerability ID: CLN-005

Vulnerability type: Vulnerable and Outdated Components

Threat level: Low

Description:

Karrot has numerous outdated python dependencies.

Technical description:

We ran a scan of Karrot using the Python `safety` scanner which noted numerous (slightly) outdated python dependencies with known vulnerabilities.

```
48 vulnerabilities found, 0 ignored due to policy.
23 fixes suggested, resolving 48 vulnerabilities.
```

We reviewed the vulnerabilities and found that none were likely to affect Karrot or warrant their own findings.

The client noted that this is due to a move from GitHub to Codeberg and a temporary loss in automated dependency management.

Impact:

- Out-of-date dependencies can contain known or unknown vulnerabilities which, even if not relevant to Karrot today, can become exploitable as Karrot's development continues.

Recommendation:

- Update all dependencies, `django` and `djangorestframework` in particular.
- Implement automated scanning and dependency management.

See also: https://owasp.org/Top10/A06_2021-Vulnerable_and_Outdated_Components/.

4.8 CLN-006 — Reserved names deny list not imposed on usernames

Vulnerability ID: CLN-006

Vulnerability type: Insecure Design

Threat level: Low

Description:

The hard-coded deny list of reserved names for user display names is not imposed on usernames.

Technical description:

Karrot prohibits users from selecting certain display names with a deny list as indicated in the following code snippet from `settings.py`:

```
# Names that shouldn't be used by groups or users because they are either confusing or unspecific
# Values are case-insensitive
RESERVED_NAMES = (
    "karrot",
    "foodsaving",
    "foodsharing",
)
```

As the comment indicates, this is to avoid users choosing a display name that could be confused with the app itself. However, this prohibition does not extend to usernames, and users can select any of these `RESERVED_NAMES` at signup.

As we noted, users can change their `username`, though this ability is unintentional CLN-002 (page 11). Through this mechanism, users are also not prohibited from changing their `username` to any of the `RESERVED_NAMES`.

Impact:

- Users can select a username from the list of `RESERVED_NAMES` leading to possible confusion.
- Users can change their username to a name from the list of `RESERVED_NAMES`.

Recommendation:

- Apply the `RESERVED_NAMES` deny list to usernames.

4.9 CLN-009 — Editors can change activity type

Vulnerability ID: CLN-009

Vulnerability type: Broken Object Property Level Authorization

Threat level: Low

Description:

It's possible to change an activity's type through the API.

Technical description:

We found that editors can change the `activity_type` property for an activity by including the change in a PATCH request to the `/api/activities/` endpoint:

```
PATCH https://dev.karrot.world/api/activities/39785/ HTTP/1.1
host: dev.karrot.world

...
Content-Disposition: form-data; name="document"; filename="blob"
Content-Type: application/json

{"id":39785,"activity_type":360}
----------------------------228323054319968193243979893991--
```

Changing an activity's type is not possible through the UI and is not otherwise an implemented feature. This has very little implication security-wise, but we include it as another Broken Object Property Level Authorization finding, similar but less consequential to CLN-002 (page 11).

Impact:

- Editors can change an activity's `activity_type` through a direct PATCH request.

Recommendation:

- Ensure the `activity_type` field is read-only.

5 Future Work

- **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.

6 Conclusion

We discovered 3 Moderate and 6 Low-severity issues during this penetration test.

Overall, we found both Karrot's frontend and backend components to be quite securely implemented. We were unable to find any typical injection-based vulnerabilities, which are common in such social applications with a large attack surface of user content – and not for lack of trying! Similarly, we found that the backend API was quite robust with well-executed access controls and authorization on every endpoint. We only found 2 minor gaps in the API's access controls, with minimal security consequences.

We did find some higher-level oversights in secure application design. A total lack of rate limiting was connected with several findings, most especially the exposure of authentication to brute-force-based attacks. When combined with a lack of a password policy, this creates a situation where brute-force attacks on authentication are not only possible but more likely to succeed. Similarly, Karrot has a very lax policy regarding file uploads with no restrictions on file types.

Karrot is designed with self-organized communities in mind, prioritizing democratic management and horizontal collaborative relationships based on trust and transparency. Despite the expected basis of trust between users, widely recommended mitigations to protect users from external attackers and potentially from each other should not be overlooked.

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

Martin Cuddy *(pentester)*	Martin has the OSCP certification and a certificate in applied cybersecurity from McGill University, Montreal, as well as an MSc. in biology from an earlier life. Besides pentesting, he is also active in promoting cybersecurity best practices for individuals, open source software, and self-hosting through workshops and writing. He's often at Montreal's Foulab hackerspace, or riding a cargo bike around the city.
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.