Radically Open Security

Penetration Test Report

Peergos

V 1.0
Amsterdam, November 18th, 2024
Confidential

Document Properties

Client	Peergos
Title	Penetration Test Report
Targets	Peergos web-ui v0.20.0, commit 8d4da5616964fb893203b636ff19e552da26fb27 Peergos web-ui v0.21.0, commit 1e7e4db16ee109389f4861406ace074878393d58 (retest)
Version	1.0
Pentester	Thomas Rinsma
Authors	Thomas Rinsma, Marcus Bointon
Reviewed by	Marcus Bointon
Approved by	Melanie Rieback

Version control

Version	Date	Author	Description
0.1	November 8th, 2024	Thomas Rinsma	Initial draft
0.2	November 12th, 2024	Marcus Bointon	Review
1.0	November 18th, 2024	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 September 23, 2024 and November 8, 2024, Radically Open Security B.V. carried out a penetration test for Peergos.

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:

- Peergos `web-ui` v0.20.0, commit `8d4da5616964fb893203b636ff19e552da26fb27`
- Peergos `web-ui` v0.21.0, commit `1e7e4db16ee109389f4861406ace074878393d58` (retest)

The scoped focus areas are broken down as follows:

- Block access control: 2 days
- Login of Peergos: 2 days
- Application sandbox: 1 days
- Secret links: 1 days
- Reporting: 1 days
- Retest: 4 hours
- **Total effort: 7.5 days**

1.3 Project objectives

ROS will perform a penetration test and code audit of Peergos in order to assess its security. To do so, ROS will audit and test Peergos locally, 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 September 23, 2024 and November 8, 2024.

1.5 Results In A Nutshell

During this crystal-box penetration test we found 2 Moderate and 6 Low-severity issues.

Peergos as a platform provides peer-to-peer file storage and sharing with a high level of privacy, even for metadata like file names and paths. This results in a unique architecture, where such information is encrypted client-side before being transmitted to a Peergos server and further into the IPFS peer-to-peer network.

Because of this, it is quite important that user-controlled data is properly sanitized and validated on the client-side after it is retrieved from the server. Even if the core cryptographic protocols underlying Peergos are secure, vulnerabilities in the front-end could undermine the overall security. Findings PRG-004 (page 17) and PRG-007 (page 11) are examples of this. Luckily, Peergos implements several modern countermeasures (frame sandboxing and a strict Content-Security Policy (CSP)) which prevent malicious script execution in the case of HTML injection, generally reducing the impact of such issues to denial-of-service at worst.

A different but related group of vulnerabilities is formed by PRG-003 (page 15), PRG-013 (page 13), and PRG-014 (page 21). These findings all relate to JavaScript-specific oddities. Specifically, a lack of data or type validation leads to situations where default JavaScript object properties can be overwritten, leading to exceptions which can render aspects of the Peergos UI non-functional.

By abusing the moderate-severity findings, an attacker could permanently disable core Peergos functionality for a victim account. The only requirement is that the victim accepts an invitation for a follower, shared folder, or shared calendar appointment.

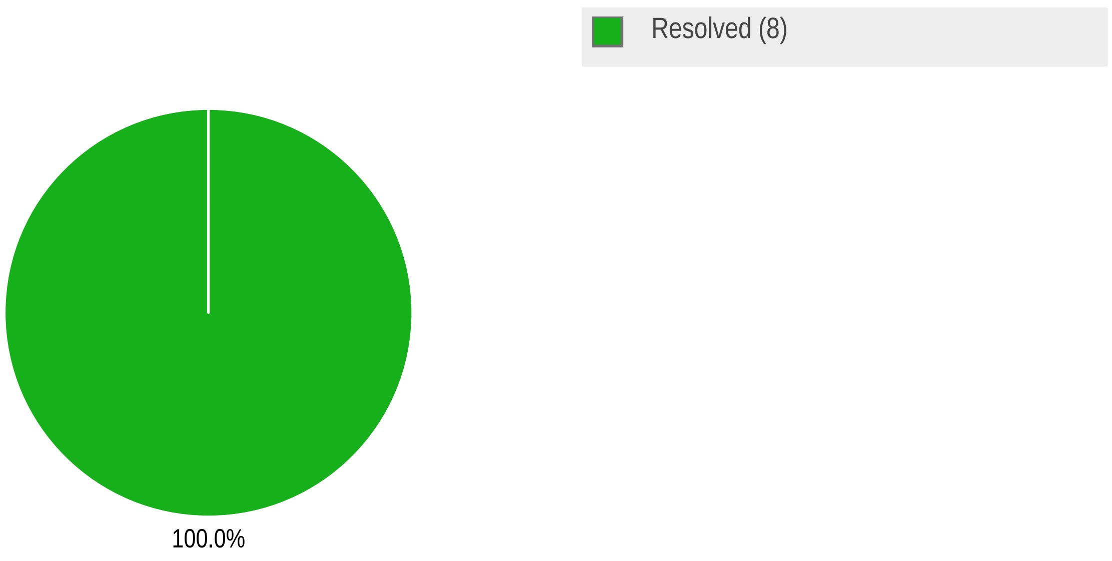
The remaining findings are not easily grouped, and all pose a low risk. We recommend following the suggested solutions provided in each finding.

1.6 Summary of Findings

ID	Type	Description	Threat level
PRG-007	Denial of Service	HTML content is not properly filtered or escaped in the calendar appointment subject and location. Through DOM clobbering, this breaks the Calendar application when these fields are rendered (in the overview and the detail popup respectively).	Moderate
PRG-013	Denial of Service	When an attacker sends a follow invitation to a victim from an account named constructor, this will trigger an exception when the victim accepts the invitation, permanently crashing their Social page afterwards.	Moderate
PRG-002	Path Traversal	When Peergos is deployed using the -webroot development flag, it is possible to fetch arbitrary files from the server's filesystem, including those outside the webroot folder, by making use of path traversal (../).	Low
PRG-003	Denial of Service	When viewing a folder (in the default grid view) containing a file with a name like constructor or toString, the Vue2	Low

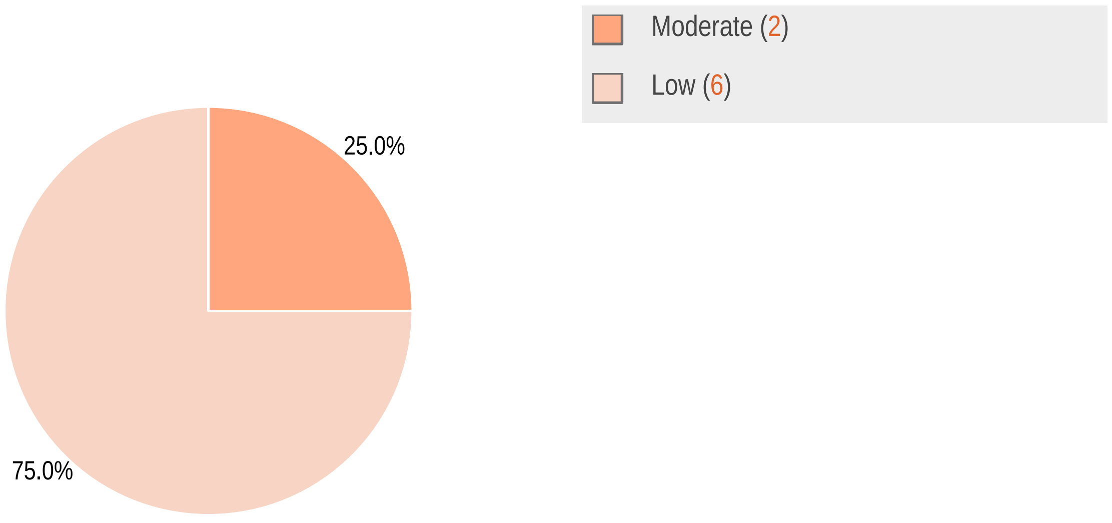
		rendering code throws an uncaught exception, breaking the entire Peergos UI.	
PRG-004	Denial of Service	By editing the front-end code to bypass the validation logic upon file/folder creation, it is possible to include the forbidden character / in file- and folder names. When viewing a folder, various pieces of path-related logic still assume a "safe" filename, but this is not checked again. The result is that some front-end logic breaks when viewing the file/folder.	Low
PRG-012	Unsafe Deserialization	The WebAuthn.Verifier class uses Java's ObjectOutputStream and ObjectInputStream for serialization and deserialization of the WebAuthn state in the database. Due to a type confusion bug, it is possible to invoke the deserialization on random bytes instead of a valid object, resulting in unpredictable behavior.	Low
PRG-014	Denial of Service	App metadata is not thoroughly type-checked, leading to a minor denial-of-service attack in the Launcher.	Low
PRG-015	HTML Injection	For its print preview functionality, Peergos makes use of DOMPurify to prevent HTTP leaks. However, the version of DOMPurify used is outdated and contains several known vulnerabilities.	Low

1.6.1 Retest Status

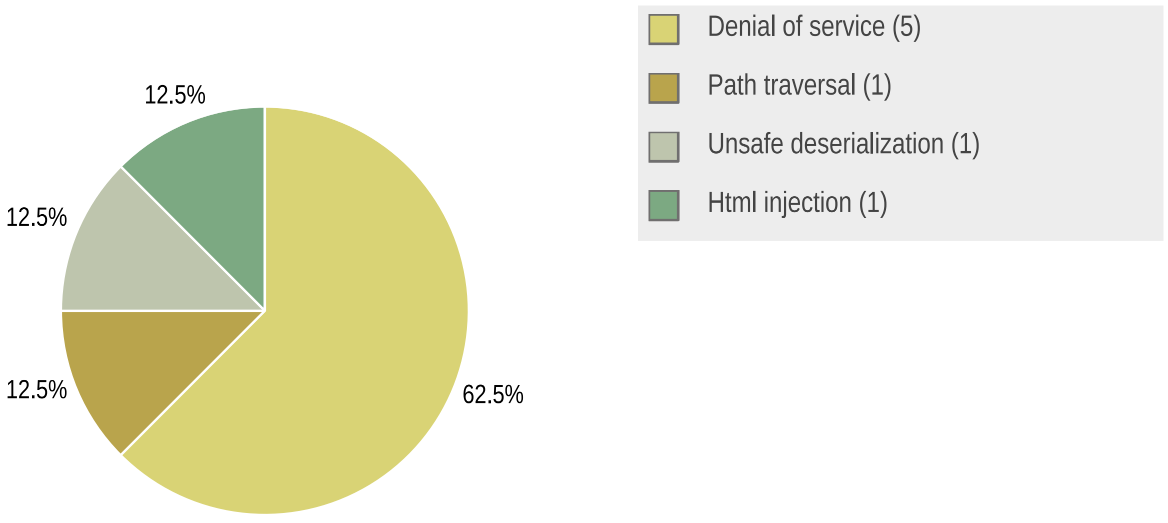


1.6.2 Distribution of the resolved findings

1.6.2.1 Findings by risk classification



1.6.2.2 Findings by type



1.7 Summary of Recommendations

ID	Type	Recommendation
PRG-007	Denial of Service	• Sanitize all inputs and escape all outputs appropriately.
PRG-013	Denial of Service	• Prevent the use of `constructor` as a username. • Upgrade to a newer, supported version of Vue.
PRG-002	Path Traversal	• Resolve the true path and validate that the requester is allowed to access files in that path before returning file contents.
PRG-003	Denial of Service	• Upgrade to a newer, supported version of Vue that does not contain this problem.
PRG-004	Denial of Service	• Ensure that filenames are validated the same way everywhere, and deny the use of `/` in filenames.
PRG-012	Unsafe Deserialization	• Take the `MultiFactorAuthMethod` from the database entry instead of the input parameters.
PRG-014	Denial of Service	• Define a strict data-format for the application manifest object.
PRG-015	HTML Injection	• Update DOMPurify, and keep it up to date in future.

2 Methodology

2.1 Planning

Our general approach during penetration tests is as follows:

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

2.2 Risk Classification

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

These categories are:

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

- **High**
  High risk of security controls being compromised with the potential for significant financial/reputational losses occurring as a result.
- **Elevated**
  Elevated risk of security controls being compromised with the potential for material financial/reputational losses occurring as a result.
- **Moderate**
  Moderate risk of security controls being compromised with the potential for limited financial/reputational losses occurring as a result.
- **Low**
  Low risk of security controls being compromised with measurable negative impacts as a result.

3 Findings

We have identified the following issues:

3.1 PRG-007 — Malicious calendar event can break Calendar app

Vulnerability ID: PRG-007	**Status:** Resolved
Vulnerability type: Denial of Service	
Threat level: Moderate	

Description:

HTML content is not properly filtered or escaped in the calendar appointment subject and location. Through DOM clobbering, this breaks the Calendar application when these fields are rendered (in the overview and the detail popup respectively).

Technical description:

On the main overview page of the Calendar app (showing a daily, weekly or monthly overview), the `subject` field of each appointment is rendered, including any HTML they contain. When an appointment is clicked, a detail dialog is shown where all the appointment's fields are visible. There, the `subject` field is printed securely, but the `location` field is not.

Due to various restrictions, the technical impact of these two HTML-injection sinks is limited:

- A strong Content-Security Policy (CSP) prevents all external requests, so it's not possible to leak the fact that the Calendar app, or a specific appointment, was opened.
- The CSP also blocks inline scripts, so Cross-Site Scripting (XSS) attacks are mitigated.
- The HTML is inserted at runtime via `innerHTML`, which means that it is mutated by the browser to be somewhat valid by itself. This mitigates defacement-attacks through closing parent tags.

However, it is still possible to do the following:

- Inserting tags with custom CSS can break the visual layout of the page (e.g., by overlaying a big element on top of others).
- Using DOM clobbering techniques (through `id` and `name` attributes of inserted tags) to manipulate scripts running on the page.

The latter attack seems to have the worst impact. Specifically, the following payload can be used to clobber the method `document.getElementById`:

```
<embed name="getElementById">
```

which will cause an exception to be thrown when the Calendar app is loaded, as code in `calendar/init.js` makes heavy use of that function:

```
init.js:1906 Uncaught TypeError: document.getElementById is not a function
    at setRenderRangeText (init.js:1906:30)
    at load (init.js:1365:5)
    at handler (init.js:23:11)
```

As with similar bugs, such an exception causes a permanent "spinner" animation to overlay the page, rendering the Calendar app unusable.

Impact:

Concretely, an attacker can abuse this vulnerability in one of two different ways:

1. The attacker creates a malicious `.ics` file and sends it to the victim out-of-band (e.g., via email). The victim imports the file in their Peergos Calendar.
2. The attacker creates a shared Calendar in Peergos and shares it with the victim's account. The victim accepts the offer, importing it into their Calendar app.

In both scenarios, the Calendar app will immediately crash when opened. This also disables the interfaces for removing shared calendars and calendar appointments, making it impossible to restore normal functionality without manual effort using browser's developer tools.

Recommendation:

Sanitize such attacker-controlled fields (i.e., all calendar appointment fields) on input, and escape them appropriately before showing them. Alternatively, different methods can be used to insert the data into the DOM, such as using `innerText` instead of `innerHTML`.

Update :

While this finding has been resolved in terms of its main impact (DOM clobbering is prevented), it's still possible to insert images and elements with custom CSS, breaking the page layout. We recommend fully escaping inserted HTML instead, to prevent this.

3.2 PRG-013 — Permanently broken Social page after accepting invitation from a specially crafted username

Vulnerability ID: PRG-013	**Status:** Resolved
Vulnerability type: Denial of Service	
Threat level: Moderate	

Description:

When an attacker sends a follow invitation to a victim from an account named `constructor`, this will trigger an exception when the victim accepts the invitation, permanently crashing their Social page afterwards.

Technical description:

The steps to perform this attack are as follows:

1. The attacker creates a user account called `constructor`.
2. The attacker sends a follow request to the victim's account.
3. The victim approves the follow request.
4. The victim can no longer load their "Social" page due to a Vue exception when it is rendered.

At step 4, the browser console contains the following exception:

```
peergos.js:14789 [Vue warn]: Error in render: "TypeError: annotations.isVerified is not a function"

found in

---> <Social>
       <App>
         <Root>
warn @ vendor.js:2690
logError @ vendor.js:3958
globalHandleError @ vendor.js:3953
handleError @ vendor.js:3913
Vue._render @ vendor.js:5630
updateComponent @ vendor.js:6138
get @ vendor.js:6550
run @ vendor.js:6625
flushSchedulerQueue @ vendor.js:6383
(anonymous) @ vendor.js:4054
flushCallbacks @ vendor.js:3980
Promise.then (async)
timerFunc @ vendor.js:4007
<... trimmed ...>
```

It is unclear exactly how the exception is triggered, but it seems to be similar to issue PRG-003 (page 15): internal logic in Vue2 which does not account for pre-existing keys in a Javascript Object which is used as a lookup-table. Due to the nature of Vue, an uncaught exception like this stops the drawing process and renders the whole UI unusable. Specifically, the Social page is not displayed properly anymore, and the victim is therefore unable to actually see their lists of friends and followers. More critically, this also makes it impossible to remove the follower that causes the crash.

The only username that can trigger this is `constructor`, as all other default `Object` properties include characters which are not allowed in Peergos usernames (i.e., uppercase letters and `_`).

Impact:

This attack is relatively simple to perform, given that the `constructor` username is still available. Once performed, the victim loses access to all the follower- and friend-management functionality on the Social page (no longer being able to see, add and remove friends/followers), which can have a large impact on their Peergos experience.

Recommendation:

Add `constructor` to the username blocklist, making this attack impossible to execute. Additionally, as a defense-in-depth measure against similar attacks, we recommend updating to Vue v3, which should mitigate issues like this.

3.3 PRG-002 — Arbitrary file read through path traversal in FileHandler

Vulnerability ID: PRG-002	**Status:** Resolved
Vulnerability type: Path Traversal	
Threat level: Low	

Description:

When Peergos is deployed using the `-webroot` development flag, it is possible to fetch arbitrary files from the server's filesystem, including those outside the webroot folder, by making use of path traversal (`../`).

Technical description:

When a directory webroot is used (the `-webroot` parameter is set), Peergos uses the `FileHandler` class to retrieve static files. Its `getAsset` method simply takes a path to a file as an argument and returns that file's contents. The issue is that `getAsset` is called from the `StaticHandler` with a path that is taken unaltered from the HTTP request URL. That means that it is possible to request a URL containing `../` and escape the webroot folder. For example, the following request will return the content of `/etc/passwd`:

```
~$ curl -s --compressed --path-as-is 'http://localhost:8000/../../../../../../etc/passwd' | head
```

```
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/usr/sbin/nologin
man:x:6:12:man:/var/cache/man:/usr/sbin/nologin
lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin
mail:x:8:8:mail:/var/mail:/usr/sbin/nologin
news:x:9:9:news:/var/spool/news:/usr/sbin/nologin
```

Note that the `-webroot` flag is not documented, and normally intended only for local UI development.

Impact:

The impact of a vulnerability like this in production would be quite high, but this directory-based webroot configuration is usually only intended for local UI development and debugging. A normal deployment makes uses of a `jar` file containing the webroot, which mitigates this issue, as (1) such path traversal is not possible within Java resource paths, and (2) it is not possible to "escape" the `jar` file anyway.

Recommendation:

While this issue only occurs in the nonstandard case where the `-webroot` parameter is supplied, which is generally not the case in production, we still recommend fixing this, as there may be legitimate production-like use cases which make use of this configuration. A mitigation could be to strip occurrences of `../` (and equivalent on Windows) from the `resourcePath`.

Alternatively, it may be beneficial to clearly document the dangers associated with using this parameter.

3.4 PRG-003 — Vue crash in Drive when a filename matches a default JavaScript Object property

Vulnerability ID: PRG-003	**Status:** Resolved
Vulnerability type: Denial of Service	
Threat level: Low	

Description:

When viewing a folder (in the default grid view) containing a file with a name like `constructor` or `toString`, the Vue2 rendering code throws an uncaught exception, breaking the entire Peergos UI.

Technical description:

The default JavaScript Object prototype (i.e., for objects constructed with `{}`) contains the following properties:

```
> Object.getOwnPropertyNames(Object.prototype)
[
  'constructor',
  '__defineGetter__',
  '__defineSetter__',
  'hasOwnProperty',
  '__lookupGetter__',
  '__lookupSetter__',
  'isPrototypeOf',
  'propertyIsEnumerable',
  'toString',
  'valueOf',
  '__proto__',
  'toLocaleString'
]
```

Due to a bug in Vue.js v2, an exception is thrown when a file in Peergos is given a name matching any of these values:

```
[Vue warn]: Duplicate keys detected: 'toString'. This may cause an update error.
found in
---> <Drive>
       <App>
         <Root> vendor.js:2690:17
[Vue warn]: Error in nextTick: "TypeError: a is undefined" vendor.js:2690:17
TypeError: a is undefined
    sameVnode http://localhost:8000/js/vendor.js:7912
    updateChildren http://localhost:8000/js/vendor.js:8314
    patchVnode http://localhost:8000/js/vendor.js:8414
    patch http://localhost:8000/js/vendor.js:8575
    _update http://localhost:8000/js/vendor.js:6020
    updateComponent http://localhost:8000/js/vendor.js:6138
    get http://localhost:8000/js/vendor.js:6550
    run http://localhost:8000/js/vendor.js:6625
    flushSchedulerQueue http://localhost:8000/js/vendor.js:6383
    nextTick http://localhost:8000/js/vendor.js:4054
    flushCallbacks http://localhost:8000/js/vendor.js:3980
    promise callback*timerFunc http://localhost:8000/js/vendor.js:4007
    nextTick http://localhost:8000/js/vendor.js:4064
    queueWatcher http://localhost:8000/js/vendor.js:6475
    update http://localhost:8000/js/vendor.js:6615
    notify http://localhost:8000/js/vendor.js:2801
    reactiveSetter http://localhost:8000/js/vendor.js:3126
    proxySetter http://localhost:8000/js/vendor.js:6699
    updateFiles http://localhost:8000/js/peergos.js:17053
```

The problem appears to be that somewhere in Vue's implementation for reactive values, object values are (temporarily) stored and/or retrieved as keys in an Object. A clash then occurs when a value like `toString` is checked, as it appears to exist, but it contains a `Function` as opposed to the expected Vue-internal data structure. This issue on the Vue v2 issue tracker appears to involve the same underlying problem. Due to the deprecation of v2, this was never resolved.

Impact:

As the Vue exception is not caught properly, it almost entirely bricks the Peergos front-end as Vue no longer properly responds to events such as navigation. A manual page refresh is required to resolve this, however, this will re-trigger the crash if the same folder is viewed.

An attacker could exploit this by creating a file with such a name inside a shared folder. When the victim attempts to list the containing folder, their page will crash. Luckily, the list view is not affected, so by switching to that, the malicious file can be deleted.

Recommendation:

As the problem is internal to Vue v2, there is not a simple fix that can be made in Peergos itself. Hence, we recommend upgrading Vue to a more up-to-date version, especially as v2 is deprecated and may therefore contain other (security-relevant) bugs as well.

3.5 PRG-004 — Inconsistent filename validation leads to potential DoS

Vulnerability ID: PRG-004	**Status:** Resolved
Vulnerability type: Denial of Service	
Threat level: Low	

Description:

By editing the front-end code to bypass the validation logic upon file/folder creation, it is possible to include the forbidden character `/` in file- and folder names. When viewing a folder, various pieces of path-related logic still assume a "safe" filename, but this is not checked again. The result is that some front-end logic breaks when viewing the file/folder.

Technical description:

The `FileWrapper` class contains a method `isLegalName`, which is used to validate file and folder names:

```
static boolean isLegalName(String name) {
        return !name.contains("/") && ! name.equals(".") && ! name.equals("..") && ! name.isEmpty();
    }
```

This version of the validation logic is invoked (among other places) during file upload and directory creation. Notably, all of this happens on the client-side, as filenames are encrypted prior to submission to the back-end. The problem occurs because other front-end logic which relates to parsing, viewing and rendering files does not always verify filenames in the same manner, and when it does, the result is not a graceful error.

For example, the `PathElement` constructor does check for `/`, but it throws an `IllegalStateException` which may not always be caught gracefully.

```
public PathElement(String name) {
    if (name.contains("/") || name.length() > FileProperties.MAX_FILE_NAME_SIZE)
        throw new IllegalStateException("Invalid path element");
    this.name = name;
}
```

The `FileProperties` constructor also performs some validation on filenames, but it does not check for the illegal `/` character:

```
public FileProperties(String name,
                      boolean isDirectory,
                      boolean isLink,
                      String mimeType,
                      int sizeHi, int sizeLo,
                      LocalDateTime modified,
                      LocalDateTime created,
                      boolean isHidden,
                      Optional<Thumbnail> thumbnail,
                      Optional<byte[]> streamSecret) {
    if (name.length() > MAX_FILE_NAME_SIZE)
        throw new IllegalStateException("File and directory names must be less than 256
 characters.");

    ... <trimmed> ...
}
```

Hence, if an attacker bypasses the validation logic during file (or folder) creation, the file is stored normally. If it's inside a shared folder, other users may attempt to list the file, resulting in various forms of broken UI functionality.

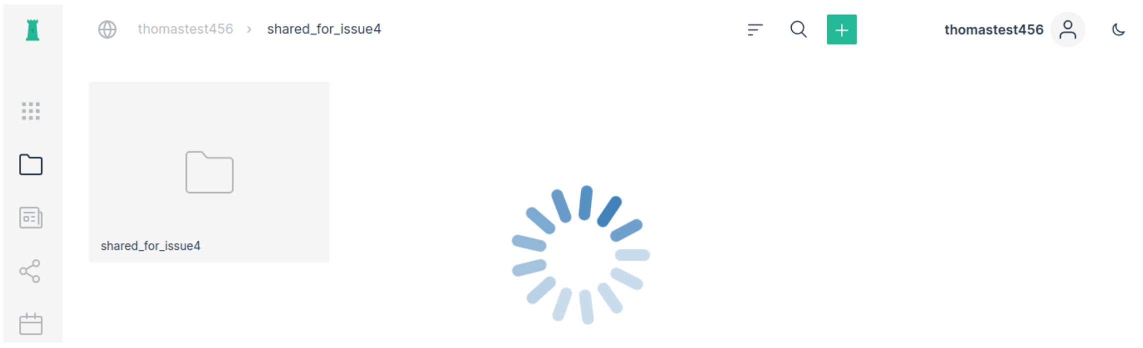
Note that we did not exhaustively check every code-flow involving filename validation; the above instances are just examples.

Impact:

The exact impact varies, as there are several code-paths that may break depending on which user is the attacker in a shared-folder scenario.

When a file or folder with the name `foo/bar` is created somewhere inside a shared folder, the following happens depending on the exact situation:

- When the victim is the owner of the folder: attempting to navigate to the parent directory of the malicious file/folder causes an infinite loading screen (spinning circle), but it's still possible to navigate away as Drive/Peergos buttons remain usable.



- When the victim is a person having received access to the folder: its parent directory will appear empty, also hiding all other legitimate files and sub-folders inside that folder.

In both cases there is a minor DoS impact. Arguably, in a writable shared-folder scenario there is already a level of trust between the parties, but this situation is a bit more conducive to abuse, as it is difficult for the victim to remove the malicious file/folder without also removing the entire parent directory.

Recommendation:

Align data validation in both directions, i.e., when submitting or modifying data on the one hand, and when retrieving data on the other hand. For data which is shared between different users (like filenames) this is of particular importance as the initial validation can easily be bypassed on the client side.

More specifically for this scenario, it is also important to consider the impact of an error condition on overall functionality of the application: a broken filename should not impact other files or folders.

3.6 PRG-012 — Unsafe deserialization of random bytes as Java object

Vulnerability ID: PRG-012	**Status:** Resolved
Vulnerability type: Unsafe Deserialization	
Threat level: Low	

Description:

The `WebAuthn.Verifier` class uses Java's `ObjectOutputStream` and `ObjectInputStream` for serialization and deserialization of the WebAuthn state in the database. Due to a type confusion bug, it is possible to invoke the deserialization on random bytes instead of a valid object, resulting in unpredictable behavior.

Technical description:

Peergos supports two forms of two-factor authentication: TOTP-based, and WebAuthn-based. This is defined in `MultiFactorAuthMethod`:

```
public class MultiFactorAuthMethod implements Cborable {

    private static Map<Integer, MultiFactorAuthMethod.Type> byValue = new HashMap<>();

    @JsType
    public enum Type {
        TOTP(0x1, false),
        WEBAUTHN(0x2, true);

        <trimmed>
    }
    <trimmed>
}
```

To keep track of this, the database schema for the `mfa` table contains a `type` column. The `value` column contains the TOTP secret in the case of `Type.TOTP`, and a serialized form of a `WebAuthn.Verifier` object in case of `Type.WEBAUTH`.

The `WebAuthn.Verifier` class implements the `Authenticator` class provided by the dependency webauthn4j. Upon each authentication event, the instance of this object is deserialized, updated (a counter is incremented), and serialized again.

The problem in this logic is that the `MultiFactorAuthMethod` to be used is specified by the user in the authentication attempt, together with an identifier of an entry in the `mfa` database table. This means that a user can ask the back-end to attempt WebAuthn authentication on a TOTP-type entry. In this case, the `value` field contains a random 32-byte string (the TOTP secret), which is deserialized using `ObjectInputStream`.

Impact:

Because the TOTP secret is not attacker-controllable, the input to `ObjectInputStream` is purely random. At worst, an attacker can repeatedly generate new TOTP secrets and exploit this type confusion bug. Some of these random strings may result in valid Java objects, resulting in unpredictable behavior. Primarily this will be in the form of exceptions which will be caught by the request handler further up the call-stack, hence the overall impact is low.

Recommendation:

By taking the `MultiFactorAuthMethod` from the database entry instead of the input parameters, the type confusion issue would be solved. As an extra layer of defense, we recommend moving away from `ObjectInputStream`, and manually implementing the (de)serialization process instead.

3.7 PRG-014 — Improper type validation for fields in app manifest JSON

Vulnerability ID: PRG-014	**Status:** Resolved
Vulnerability type: Denial of Service	
Threat level: Low	

Description:

App metadata is not thoroughly type-checked, leading to a minor denial-of-service attack in the Launcher.

Technical description:

Peergos apps specify their metadata through a `peergos-app.json` file. During app installation, this file is read and verified by the `verifyJSONFile` method. While most fields are parsed thoroughly and correctly, some others are only superficially checked. Specifically, there are no type checks for several fields where strings are expected.

The `author` and `description` fields both have length limits:

```
if (props.description.length > 100) {
  errors.push("Invalid description property. Length must not exceed 100 characters");
}
...
if (props.author.length > 32) {
  errors.push("Invalid Author property. Length must not exceed 32 characters");
}
```

However, without a proper type-check, this can be bypassed by passing a JSON (and hence JavaScript) object or array instead. To demonstrate:

```
let example = ["long string here, AAAA<trimmed>"]; // example.length == 1
let example = {"foo":"long string here, AAAA<trimmed>", "length": 31}; // example.length == 31
```

In both cases, such values will pass the length checks, but when they're later (implicitly) cast to a string for display purposes, the resulting string can be arbitrarily long. Additionally, when an object is provided that includes the key `toString`, the app installation dialog will not show as the implicit string-conversion will throw an exception.

The `source` field behaves similarly, but there is no attempt to show it on the app installation dialog. It is however rendered on the "details" page in the Launcher, and also during the process triggered by the "Check for updates" button in the Launcher. Hence, setting `source` to e.g. `{"toString":"foo"}` will allow the application to install correctly, but break both of the above-mentioned processes.

Impact:

A malicious app developer could cause a minor denial-of-service attack by setting the `source` field in their application manifest to `{"toString":"foo"}`. When this application is installed, the user will no longer be able to use the "Check for updates" functionality, nor see the malicious application's "details" dialog.

Recommendation:

Define a strict data-format for the application manifest object. This can be done by thoroughly (manually) type-checking each of the fields, or by using a data validation library such as Zod.

3.8 PRG-015 — Outdated version of DOMPurify may be vulnerable to bypasses

Vulnerability ID: PRG-015	**Status:** Resolved
Vulnerability type: HTML Injection	
Threat level: Low	

Description:

For its *print preview* functionality, Peergos makes use of DOMPurify to prevent HTTP leaks. However, the version of DOMPurify used is outdated and contains several known vulnerabilities.

Technical description:

Peergos makes use of DOMPurify version 2.3.6 from February 2022. Since then, several vulnerabilities have been fixed. For some of these there are publicly available exploits, but not for all. Such exploits abuse differences in parsing (and mutation) between the browser and DOMPurify, resulting in the ability to inject unfiltered HTML code.

Below are relevant DOMPurify releases since 2.3.6 that fix (potential) bypasses:

- Fixed prototype pollution issue: https://github.com/cure53/DOMPurify/releases/tag/2.4.2
- Fixed two bypasses: https://github.com/cure53/DOMPurify/releases/tag/2.4.8
- Fixed a bypass: https://github.com/cure53/DOMPurify/releases/tag/2.4.9
- Fixed an mXSS-based bypass: https://github.com/cure53/DOMPurify/releases/tag/2.5.1
- Fixed two mXSS variants: https://github.com/cure53/DOMPurify/releases/tag/2.5.2

- Fixed several mXSS variations: https://github.com/cure53/DOMPurify/releases/tag/2.5.3
- Fixed possible bypasses: https://github.com/cure53/DOMPurify/releases/tag/2.5.7

Impact:

Luckily, it appears that the publicly available bypasses for this version all make use of `<svg>` tags, which happen to be blocked (for a different reason) in Peergos' usage. Hence, we were unable to inject HTML code and bypass the HTTP leak filter.

Recommendation:

Even though we were unable to bypass the filter during this pentest, there may be other exploits for this version of DOMPurify (e.g., privately-known or reverse-engineered from the changes) that do not rely on `<svg>` and that would work in this scenario. Consequently, we strongly recommend updating DOMPurify, and keeping it up-to-date in the future.

4 Non-Findings

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

4.1 NF-001 — Old PDF.js version is vulnerable to JavaScript injection

Peergos makes use of the `PDF.js` library to display PDF content. The version used is v2.1.266, which is relatively old, and vulnerable to CVE-2024-4367, which allows for arbitrary JavaScript code execution through a malicious PDF file. However, due to the Content Security Policy and sandboxing features that are in place, such code execution attempts are blocked, so there is no impact.

UPDATE: in a retest, we noted that the included distribution of `PDF.js` has been updated to the latest version.

4.2 NF-006 — Validation flaws in ICS import

The JavaScript based ICS importer (`ics.js`) does not always properly validate field contents, and may throw an exception on malformed data.

For example, by setting an appointment's `UID` field to `__proto__`, the import process throws an exception. Because this has no further effect besides breaking imports, there is no security impact.

4.3 NF-008 — HTML injection via calendar color

Each individual calendar in the Calendar app has an associated `color` property. This field is used as a CSS color value to style various aspects of the calendar, for example in `replaceCalendarsInUI()` as `calendar.borderColor`, used for drawing a filled circle:

```
function replaceCalendarsInUI() {
    var calendarList = document.getElementById('calendarList');
    var html = [];
    CalendarList.forEach(function(calendar) {
            html.push('<div class="lnb-calendars-item"><label>' +
                '<input type="checkbox" class="tui-full-calendar-checkbox-round" value="' +
 calendar.id + '" checked>' +
                '<span style="border-color: ' + calendar.borderColor + '; background-color: ' +
 calendar.borderColor + ';"></span>' +
                '</label><label><span id="cal-' + calendar.id + '" style="cursor:text;"></span>' +
                '</label></div>'
            );
    });
    <trimmed>
}
```

By manually modifying the value as it is being changed using the color picker, it is possible to set this value to an arbitrary string. This allows for a full HTML injection by escaping the CSS context. For example, with the following payload:

```
#FF0000;"><b>hello</b>
```

This will persistently show the injected HTML next to the colored circle, every time the Calendar app is opened. There is however no security impact here, as this field is local to the account, and not coupled to a calendar when it is shared between users. Hence, it is not possible to share this payload with a different user account.

4.4 NF-010 — Crafted message sent from an app triggers an exception in the Service Worker

By sending a crafted message from within an app, an exception can be triggered in the Service Worker (`sw.js`).

The following snippet will sends a message with a `filename` object where a string is expected:

```
window.top.frames[0].postMessage({filename: {}}, '*', [(new MessageChannel()).port1])
```

This causes the following exception:

```
TypeError: filename.startsWith is not a function
    at sw.js:21:16
```

There is no security impact, as the Service Worker continues functioning normally.

5 Future Work

- **Testing the IPFS implementation**
  The core implementation of the peer-to-peer networking functionality used by Peergos was not in scope for this security evaluation. Given that there is a potential risk of network-scale attacks at this layer, it may be worth it to look into this further.
- **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 2 Moderate and 6 Low-severity issues during this penetration test.

Overall, we find that Peergos is relatively strong architecturally, i.e., in the aspects of encryption, authentication, and access control. This includes its relatively new *block access control* feature. We also did not find any concrete problems with its application sandbox: due to the usage of modern browser security mechanisms such as a strong Content-Security Policy (CSP), the provided privacy guarantee holds true.

The problems that *were* found however, show that more focus could be placed on data validation and sanitization on the client-side. While no Cross-Site Scripting (XSS) attacks were possible, we found several ways to exploit these instances of poor validation, resulting in a targeted denial-of-service attack, breaking core Peergos UI functionality for victim accounts.

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

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