Security Audit of Seafile’s Private Cloud Storage platform

UNIVERSITY OF AMSTERDAM

Offensive Technologies Report

Jeroen Schutrup
jeroen.schutrup@os3.nl

Siem Hermans
siem.hermans@os3.nl

May 27, 2016

Abstract

Public cloud storage platforms offer a great deal of flexibility which has caused many companies to include this paradigm in their IT strategy. However, as the dependence of companies on cloud storage rises, concerns regarding the safety of these platforms emerge. As such, an increasing amount of companies is investing in privately hosted storage solutions like Seafile. Still, due to the public interfaces provided by these products, potential security vulnerabilities remain present. This paper presents a security audit of Seafile’s private cloud storage solution. It identifies common attack vectors and vulnerabilities of comparable products and examines the exploitability of Seafile in its current state. All identified vulnerabilities are documented and weighted to indicate their effective risk. Lastly, advice is given for future improvements of the platform with regards to security. Although Seafile as a platform has made great advancements regarding security in recent years, this audit reveals several high risk vulnerabilities ranging from denial of service exploits to the ability to gain unprivileged access to a wide range of stored data. Moving forward, it is advised to perform a thorough revision of the current code base to eliminate the uncovered vulnerabilities.

Keywords – Private Cloud Storage, Security Audit, vulnerabilities, Seafile

1 Introduction

According to predictions of Gartner in 2012 [1], a third of all data would be stored in the cloud by 2016. This trend has generally held true as more and more companies started adopting cloud storage platforms as either a backup for their data or even as a primary storage solution. Cloud storage has rapidly evolved from being a promising business concept to a mainstream strategy. However, as with any publicly provided third party service, these platforms are potentially vulnerable to security incidents [2], data breaches [3], and other malicious activities. This means that as of right now, many companies are still anxious to migrate their data due to sovereignty, security and privacy issues [4, 5]. These concerns have caused an influx in the popularity of privately hosted, on-premise cloud solutions which essentially allow a company to provide similar services as commercial platforms like Dropbox and Google Drive, but selectively to a group of users whilst maintaining centralized control over the data. As such, data privacy and protection can be maintained. Prime examples of platforms which provide these functionalities nowadays are ownCloud and Seafile. Most notably, ownCloud [6] has been gaining a lot of traction and sees an increasing amount of production deployments [7]. However, similar competing products like Seafile are up and coming.

This new interest in private cloud solutions raises questions regarding the effective security of these platforms. Although, by using a private cloud storage platform, corporate data is not stored with a third party anymore, the data is still commonly being provided to end users via a series of public interfaces. The security of ownCloud, as a result of its popularity, has been researched in the past multiple times. However, at the time of writing we are not aware of any security audits with regard to Seafile. This is alarming, as a series of serious security flaws have been reported in the past years regarding the platform [8]. These issues concern Initialization Vector (IV) reuse in Seafile’s data encryption and bad password salting. As such we are interested in the exploitability of the freely avail-
able Community Edition of Seafile’s private cloud storage platform.

This report aims to present a broad yet complete security audit of the web access interface of Seafile’s private cloud storage solution. Prior to the audit we define a pragmatic approach which aims to identify the most successful areas of discovering a vulnerability. In the process we take a look at recent security flaws reported for private cloud solutions in general and relate relevant issues to Seafile. The following contributions have been identified:

- Exploration of common and recent security flaws regarding popular private cloud storage solutions.
- Presentation of identified security flaws regarding Seafile specifically accompanied with a risk assessment and exploitability evaluation.
- Advices regarding vulnerability mitigation and a general security advice regarding Seafile moving forward.

2 Research Questions

The aim of this project is to provide an answer to the following main research question:

What is the current state of security regarding Seafile’s private cloud storage solution?

Several sub-questions have been posed to limit the scope of this research:

- Are vulnerability trends of major competing private cloud solutions like ownCloud applicable to Seafile?
- What is the status of security vulnerabilities found in the past and what are currently known vulnerabilities?

Further scoping of the project is discussed in more detail in section 6.

3 Related Work

The security of cloud environments in general has been researched in the past multiple times. Following the popularity of these storage as a service (StaaS) platforms, a large series of vulnerabilities are being uncovered and the amount of viable attack vectors are increasing. In order to structure these vulnerabilities, Dotter et al. [9] present a cloud platform attack and risk assessment taxonomy specifically catered towards cloud platforms. Their taxonomy divides cloud attacks by identifying the source of the attack, the attack vector, the target, corporate impact, and potential defense measures against the attack. This classification is largely similar to the slimmed down risk assessment method of the Open Web Application Security Project (OWASP) in which vulnerabilities are identified and prioritized following a standard format. Due to the positive reputation of the OWASP, any vulnerabilities uncovered during the course of this project will be assessed by taking the latter methodology as a starting point.

Competing cloud storage platforms

Due to the potential risks of storing corporate data within the public cloud, many security conscious companies are anxious to store their data with a third party. Following these concerns, private cloud storage solutions have become increasingly popular among Enterprises. As a result of its popularity, the security of ownCloud has been researched in the past multiple times. The main points of interest are usually general security concerns regarding cloud platforms or the security of the service’s web frontend. Rexha et al. [10] for example present a general overview of common security issues regarding cloud storage platforms. Conclusively they identify the client-side data synchronization functionality as the most critical attack vector. Xu et al. [11] perform a penetration test based on common vulnerabilities as identified in the OWASP Top Ten project. This project aims to identify the most critical web application security flaws of current times. Following this list they performed a series of Man in the Middle (MitM) and SQL injection attacks on an ownCloud deployment in a lab environment. Although they are successful in their MitM attempts, the success of their attacks hinges on the absence of a properly implemented certificate. Furthermore, all SQL injection attempts failed. Due to its active community and open source nature of the project, ownCloud is actively being monitored by the community which in return results in a large amount of Common Vulnerabilities and Exposures (CVE) publications.

Issue tracker & vulnerability trends

Besides the commonly researched ownCloud, Seafile is gaining in popularity. However, at the time of writing, no security evaluations of the platform have been performed. This is likely caused due to the fact that there aren’t many known production deployments. Still, interest is growing as Sheng et al. [13] propose a model for deploying Seafile in college and University environments. Their research repeatedly commends Seafile for its ‘high security’ but fails to present technical motivation as to why this level of security is reached. The issue tracker on GitHub for Seafile’s cloud solution even disputes this security claim as in the past five years 55 security issues in ranging levels of severity have been reported.
ownCloud is the prime competitor of Seafile’s platform and employs a similar concept. As such, this report inventarises the past and present vulnerabilities relevant to ownCloud. The aim is to find commonalities in vulnerabilities. Potentially these vulnerabilities can be projected on Seafile. This evaluation is presented in the following section.

4 Private cloud vulnerability trends

This section describes recently reported vulnerabilities with regards to the comparable private cloud storage solution ownCloud. By exploring these flaws we aim to discover trends in vulnerabilities for this type of platform, if any. The presented data may show promising areas for further research with regards to Seafile.

4.1 Data set

For this preliminary research the publicly available National Vulnerability Database of the National Institute of Standards and Technology (NIST)\(^\text{[14]}\) has been used. All reported vulnerabilities from 2013 onwards for ownCloud have been extracted. Subsequently this data was combined with the vulnerability severity scores of CVEDetails\(^\text{[15]}\). The following sections visualize and discuss the findings.

4.2 Vulnerabilities & severity

From 2013 onwards, ownCloud has been found prone to 87 security vulnerabilities. As presented in figure 1, the majority of these security flaws concerned Cross Site Scripting (XSS) attacks. However, further analysis in figure 2 reveals a relatively low impact regarding XSS flaws. The same holds true for attacks which attempt to bypass protection. While the least amount of vulnerabilities concerned directory traversal, these flaws do have a significant impact. Lastly, both code execution attacks and information obtaining attempts are frequently reported and share the characteristic that their impact is generally high. As these are the two form the most promising areas of research they are discussed in more detail in the section below.

4.3 Further analysis

Code execution & obtaining information

Of the fourteen remote code execution flaws in ownCloud, twelve have a low complexity and can cause considerable impact. Four of these vulnerabilities are able to completely compromise a target system. Further investigation reveals that two of these bugs are directly caused by using an external library containing the vulnerability. Another flaw is introduced by using third party extensions to the ownCloud platform. Bugs present in the product’s own code base only causes partial access.

When trying to obtain information, all but two have a low access complexity. Contrary to code execution flaws, these bugs are almost all due to inappropriate parameter checking, using bad randomness, or applying cryptography in a wrong manner. While having a considerable impact on data confidentiality most of these flaws are tough to discover.

Miscellaneous flaws

Regarding Denial of Service (DoS) vulnerabilities, nearly half of them are caused by flaws in external libraries. All remaining bugs are caused by insufficient parameter checking of which some of them are also related to directory traversal flaws.
For restriction bypasses, using external libraries or add-ons is also causing a considerable amount of security flaws.

Further research
Conclusively, vulnerabilities reported for ownCloud over the past years reveal that the usage of third party libraries and extensions generally introduces a plethora of security flaws. Secondly, the usage of bad sources for randomness or incorrectly applying cryptography (TLS verification, bad hashing algorithms) causes issues as well. Last of all, incorrect parameter checking in the web client and API of ownCloud introduced a lot of options for performing code execution attacks or allowing an attacker to obtain information. These three areas are considered to be interesting for further exploration during this research, as they are very frequently reported, have a high impact on either confidentiality, integrity or the availability of the system, while the access complexity is very low.

5 Seafile Service overview
In order to set the stage for Seafile’s platform, this section will delve briefly into the primary functions of the product and its internals. Additionally, the variety of installation methods provided by Seafile are discussed.

5.1 Core platform functionality
Due to the popularity of cloud computing as a whole, the term ‘cloud storage’ has become an overloaded and ambiguous concept which nowadays covers a broad range of technologies ranging from object storage architectures like Amazon S3 to file hosting and synchronization services like Dropbox. Seafile and ownCloud both fall into the latter category by providing a storage platform which in its core functionality can be compared to commercial products like Dropbox or Google Drive. These services are primarily catered towards enabling end users to interface with their private data in a straightforward manner. In order to facilitate this functionality, the end user can generally utilize dedicated external clients or a web interface to access the server. As is the case with Seafile. To clarify, figure 3 presents an overview of the components which make up the Seafile platform.

From a technical perspective the server consists of three primary components, namely the Seafie Daemon, the Ccnet server and the ‘Seahub’ front-end. The Seafie Daemon is responsible for handling file uploads, downloads and synchronization requests from external clients. As such, the data daemon forms the core of the platform. Additionally, this daemon allows users to create and organize ‘libraries’. These libraries function as a top level data container for each user’s personal set of files and folders. The data daemon maintains a modification history for each library. The Ccnet component is an RPC service daemon which allows for internal communication between the components. Lastly, the web interface, ‘Seahub’ forms the primary access point to the server. This component allows an end user to access their data which is stored on the centralized Seafile server. The interface is built on top of the popular Python framework ‘Django’ and by default Seahub is hosted on ‘gunicorn’: a simple Python HTTP server. As the image illustrates, dedicated (mobile) external clients are available as well. These are referenced in more detail in section 10. All of the core components rely on their own databases which are being handled by a MariaDB instance.

5.2 Installation methods
Interestingly enough, Seafile provides two distinct installation methods. The first method is a simple installation script which is included in their installation package. This script allows one to automatically deploy the previously mentioned core components of Seafile. However, Seafile also provides a separate installation script on their GitHub page which claims to automatically set up a production ready Seafile Server. The simple script merely provides guidance in installing an insecure instance of Seafile Server without HTTPS enabled. The secure version on the other hand also contains an Nginx webserver with self signed certificates and a Memchached instance which allows for better scalability of the server. In the latter instance, all requests to the Seafile environment are proxied through Nginx by running Seahub in fast-cgi mode behind the web server.

It is important to note that this security audit focuses primarily on the supposedly secure instance of Seafile, as we feel that the default instructions are mainly catered towards setting up a develop-
6 Methodology

This section discusses the methodology of the project and reviews the scoping of the audit. The assessment framework presented in this section defines the overall scope of the security audit.

6.1 Environment

To perform the audit, a privately hosted Seafile environment has been deployed in a VirtualBox virtual machine using the so-called secure installation script provided by Seafile. The machine is hosted in a private environment. During the experimentation phase, no personally identifiable data is handled. As such, no ethical concerns are present.

6.2 Assessment framework

As depicted in figure 3, a full deployment of Seafile’s Private Cloud storage environment consists of many interconnected components and extension modules. In order to effectively delimit the scope of the project, this audit will exclusively examine the server component of the platform and in particular its web front-end which allows end users to interface with their data. The preliminary research in section 4 evaluated ownCloud CVEs and identified a series of areas which are interesting for further research. Most importantly, the usage of third party libraries, bad sources for randomness and incorrect parameter checking by the application are areas of interest. Additional care will be taken when evaluating these areas.

The identified vulnerability areas are largely covered by the OWASP Top Ten. As such, this project will utilize the most recent OWASP Top Ten from 2013 as a framework for assessing the security of the platform. Doing so allows us, at a later stage, to identify the risk of any uncovered vulnerabilities following a broadly accepted scoring methodology. Listed below are the web application security risks which this audit will review:

- A1 - Injection
- A2 - Broken Authentication and Session Management
- A3 - Cross-Site Scripting (XSS)
- A4 - Insecure Direct Object References
- A5 - Security Misconfiguration
- A6 - Sensitive Data Exposure
- A7 - Missing Function Level Access Control
- A8 - Cross-Site Request Forgery (CSRF)
- A9 - Using Components with Known Vulnerabilities
- A10 - Unvalidated Redirects and Forwards

With regards to performing the actual audit, the OWASP website provides detailed guidelines and procedures for determining whether a system or application is susceptible to the security flaws listed above. Throughout the course of this audit, these guidelines will be taken into consideration whilst evaluating the Seafile platform. Additionally, this audit takes existing security vulnerabilities as reported on the GitHub issue tracker into consideration. These issues will be correlated to the vulnerabilities as identified by the OWASP as much as possible. In the past a series of reported security vulnerabilities have been closed on the Github issue tracker without any apparent solution. Therefore this audit will also describe non-results regarding these issues as they remain relevant.

Important to note is that this audit seeks to outline the security of Seafile’s Private Cloud storage platform as best as possible within the available timeframe. In principle, all the security risks listed above have been timeboxed equally. However, when interesting results were found in a particular area, more time was spent. Due to the limited time available, no definitive statement regarding the security of Seafile in any of these areas can be made. As such, all conclusions regarding the security of Seafile will be based on the findings during the audit.

7 Results

This section presents the findings as a result of performing the described method in section 6. The risks of each vulnerability will be analyzed in the second part of this section. Simultaneously, a brief demonstration is provided for the identified vulnerability which exhibits its practical applicability.

7.1 OWASP & trends

This section will describe all results with regard to OWASP and the aforementioned observed vulnerability trends.

7.1.1 A1 - Injection

The application API was tested for SQL injection vulnerabilities using SQLmap. Amongst others, the password field for opening a password-protected shared file was tested, as well as the library ID in the URI for opening a library itself. Also, the HTTP headers like the CSRF token and session ID were tested. Neither of these fields were
found vulnerable for SQL injection. Since Seafile uses Object Relational Mapping (ORM) included with the Django framework, which is used by lots of applications and has been subject by lots of pen-tests, it was very unlikely beforehand to find any vulnerabilities in this section of the application.

### 7.1.2 A2 - Authentication and Sessions

Authentication and session management is split up in a number of elements. First, password hashing and policies will be discussed, after which the subject will be changed to session management.

#### Password storage

According to GitHub issues [18, 19, 20], Seafile allegedly applied weak password hashing. Besides using SHA-1 to store user passwords, a static salt was used for every user. Most interesting however, was that passwords were hashed in the wrong way. When salting passwords, the salt should be hashed before the the password is. In prior versions of Seafile, this was done the other way around, effectively diminishing the added complexity of using a salt. In more recent versions, 1000 iterations of the PBKDF2 algorithm is used, storing a unique one-way password hash, derived from a per-user unique salt and a password as input. These elements are stored per user in the database, as depicted in table 1. Additional research is required to guarantee the cryptographic strength of this new hashing methodology. This is discussed in more detail in [10]

#### Password policy

By default, Seafile enforces so-called complex passwords. On user creation, a one time password is generated either by the Javascript `Math.random()`, or by the administrator. This password is not bound by any complexity requirements. The credentials are then mailed in clear-text to the user. On the first successful logon attempt the user is obliged to create a new password, which should consist out of at least eight characters containing at least a special-character, number and uppercase letter.

#### Session management

Session ID’s consist of 32 character alphanumeric case insensitive strings. Upon visiting the web portal unauthenticated, the server passes a Set-Cookie HTTP header to the client, containing a session ID. From this moment on, the client embeds this identifier in the Cookie HTTP header. After logging in, the session ID is rotated. Only the new session ID is usable to visit the authenticated parts of Seafile. Using the old Session ID results in a redirect to the login page. Furthermore, Session IDs have a nine day timeout period. After logging out, the Session ID is properly deactivated by the webserver.

When using the API a 64 bit hexadecimal API token is obtained using the `auth-token` API call. Based on a combination of the user, platform, and IP address the API returns an (existing) API token, depending on whether the user already obtained an API token from this platform before. If no API token for this user/platform combination exists, a SHA1 HMAC is obtained from a generated UUID as displayed in listing 1. Although revoking API tokens obtained by authenticating with mobile apps for iOS/Android is working correctly, we found no way of withdrawing API tokens that were added by using `curl`. Also, API tokens don’t have any timeout mechanism employed.

```python
    def generate_key(self):
        unique = str(uuid.uuid4())
        return hmac.new(unique, digestmod=sha1).hexdigest()
```

Listing 1: API Token generation (excerpt from auth/tokens.py)

### 7.1.3 A3 - Cross-Site Scripting

Besides storing data, Seafile allows a user to create a Wiki page within the web interface. The Wiki pages, usernames and attributes like telephone number or department, file, folder and library names, notifications and groups were extensively tested for XSS vulnerabilities. This resulted in two findings. First is the registration of new users. Although it’s disabled by default, it’s possible to register with an e-mail address like "<script>alert(1)</script>"@os3.nl. Once saved in the database, it causes the administrators to be unable to list any existing users. The issue is resolved once the user’s record is manually removed from the database.

The Wiki pages are, from a theoretical perspective also vulnerable for Cross-site Scripting. Although all HTML tags shipped with user-input is properly sanitized and escaped, it’s possible to display an image or URL pointing to an external source, using the Markdown editor for Wiki pages. However, the input ending up in the `img src` and `a href` tags is not properly escaped and therefore may include HTML tags and JavaScript code. We were not able to exploit this flaw, as the `markdown.js` plugin does not provide escaping of parentheses. This causes any entered JavaScript to miss the closing parentheses as shown in listing 2.

```
    &lt;script&gt;alert(1)&lt;/script&gt;"@os3.nl

    &lt;img src="&lt;script>alert(1)&lt;/script&gt;"@os3.nl"
```
7.1.4 A4 - Direct object referencing

Although directory traversal was not found applicable, we were able to find two API calls where parameters were not sufficiently checked. A major flaw exists enabling any authenticated user to publish arbitrary libraries to a public URL. This is caused by the share/ajax/get-download-link/ URI which performs insufficient authorization checking. In listing 3 a Proof of Concept (PoC) is demonstrated in which an authenticated user sets the repo_id to a library which doesn’t belong to the authenticated user account. The application willingly presents a download link which is open for everyone with network access to the Seafile server. From this URL, the library can be explored and any file could be downloaded. If the targeted library is encrypted, metadata like filenames are still visible in plain, and the encrypted files can be downloaded as well. This leaking of metadata information regarding encrypted libraries has been reported nearly three years ago in a GitHub issue \[8\] and remains to be fixed.

Listing 3: Sharing an arbitrary library, opening it up to the internet.

```bash
$ curl [...] https://seafile.testing.local/
   → share/ajax/get-download-link/ --data
   → "use_passwd=0&repo_id=a34c48dd-4e9f
   → -4a32-98c0-1e54ff502697&type=d
   → "

{"token": "2364921741", "download_link": 
   → "https://seafile.testing.local/d
   → /2364921741/"}
```

A second vulnerability concerned a minor flaw in the Wiki pages. After enabling the Wiki for the currently logged in user, an existing library can be selected to be used for the Wiki pages. The library ID for this request can be forged to point to another user’s library. Although editing or viewing files is not possible, a user now does have the ability to create arbitrary files in the destination library without the consent of the library’s owner.

7.1.5 A5 - Security misconfiguration

With regard to security misconfigurations this section will cover used local OS changes and application specific settings.

Installation

As discussed in section 5.5.1 Seafile is divided in several components. The Ccnet RPC daemon is merely offering functionality for local processes, hence it’s only available locally. Nginx is proxying all HTTP traffic to the Seahub daemon, which therefore doesn’t have any sockets open to the internet. Nginx correctly redirects all HTTP connection attempts to port 443. Furthermore, it also functions as a broker to the Seafile daemon through the /seafhttp URI. However, the Seafile daemon, which takes care of chunked traffic, also directly exposed by listening on port 8082. Supporting services like databases and caching daemons are only available to processes on the local host. During the installation a new user is created under which Ccnet, Seahub and Seafile are ran. This user only has write access in /opt and /var/tmp. Furthermore, the installation script generates a random password which is set for the administrator. No default passwords are enabled after the installation.

Application and configuration

Since password and session management has previously been discussed, this paragraph won’t discuss these components. Under no circumstances we found any error logging served to the client. User registration is disabled by default, so unauthenticated users are unable to post any content to the application, except for the URI obtaining an API token or session ID.

7.1.6 A6 - Sensitive data exposure

Except for using SHA1 to obfuscate API token UUID’s, Seafile uses a safe set of encryption and hashing algorithms. As discussed in section 7.1.2 passwords are stored in a safe manner. The next paragraphs will cover encryption of stored files, and subsequently file ID randomness. This section will conclude by discussing encryption used by the chunk upload/download component of Seafile.

File Storage Encryption

Seafile offers the ability to create encrypted libraries. All files in such a library will be encrypted using the same key. Upon entering the password when accessing an encrypted library, a key is generated. Also, a magic string is generated based on a concatenated string based on the library ID and the password. This will later be used to verify the entered password.

Files stored on disk are encrypted using AES-256 CBC. However, the source code also contains methods to encrypt and decrypt data using AES-128 in ECB mode for backwards compatibility, implying that there are still Seafile servers in the wild using poor encryption.

Furthermore, Seafile has allegedly been reusing Initialization Vectors (IVs) in the past \[8\]. Although the issue was reported three years ago, it’s still not fixed. Therefore, an attacker is able to leverage a known plaintext attack to guess the encryption
key. Furthermore, patterns may be discovered in encrypted data.

**Network security**
File uploads to the Seafile server are forced to perform a TLS handshake before connecting. However, the server doesn’t send an HTTP Strict Transport Security (HSTS) header, enabling an attacker to redirect requests over plain HTTP.

With regards to Nginx, the default self signed certificate which is generated when installing the Seafile server includes support for TLS_FALLBACK_SCSV. This parameter prevents a client from forcefully renegotiating weak(er) cipher suites, thus effectively preventing SSL/TLS downgrade attacks.

![Figure 4: Entropy of the share ID values as generated by Seafile](image)

**File ID randomness**
One of the areas of interest identified in section 4 was the use of bad randomness generators. When sharing files to unauthorized users, Seafile automatically generates a random URI on which the file is made available. They random string of the URI consists of 10 alphanumeric characters which theoretically means that it would be possible to traverse all shared files by repeatedly requesting the random strings. We evaluated whether this mechanism uses an invalid source of randomness. However, as depicted in figure 4, nearly all measurements are proven to be random. The dataset of shared files consists of approximately 450,000 entries. Within this amount of entries, eight double values were found. The idea is that if any patterns are visible in the plot it would not be required to traverse the whole keyspace. However, due to the low amount of double entries, no consistent pattern was determined.

7.1.7 A7 - Missing function level access control

With regard to the web interface and the API, no serious flaws were found in terms of missing access control. Also no URIs were found to leak information. Sealhub has implemented various rate limiting functionalities to prevent a large number of subsequent access attempts. The login page presents captcha’s when it detects a successive series of failed login attempts. Additionally, the API rate limits access attempts as shown in listing 4.

```bash
while read line; do curl -d "username=jschutrup@os3.nl\npassword=$line" -k https://seafile.testing.local/api2/auth/token; done <./password.list
```

Listing 4: Authentication rate limiting on web API

However, when sharing a library or file and setting a password to prevent unauthorized access, it is possible to perform a brute-force attack to the password field without any form of rate limiting. This effectively enables an unauthorized attacker to gain access to the repository given that enough resources are available.

7.1.8 A8 - Cross-Site request forgery (CSRF)
Sealhub is built on top of Django and this framework performs CSRF header checking by default for all PUT and POST requests. Although it is possible for a method to opt-out for CSRF checking, this is not the case in any of the views. No occurrences of the opt-out method were detected in the source code.

7.1.9 A9 - Using components with known vulnerabilities
Another area of interest identified in section 4 was the use of third party libraries. Contrary to ownCloud, Seafile does not feature a plug-in platform in which third party extensions can be loaded. This inherently provides some level of security. Additionally, both variants of the installation script download the latest libraries that are applicable for the provided architecture. Except for the djangorest framework, no reported vulnerabilities were found for used libraries. The djangorest v3.3.2 framework used by Seafile is one version behind, and has a reported bug concerning a missing CSRF header when using the HTML renderer for the API.
However, this renderer is not used by Seafile, limiting the exploitability for this security flaw. For all libraries, Seafile is of course vulnerable for Zero-day vulnerabilities.

7.1.10 A10 - Unvalidated redirects and forwards

We found no methods of redirecting visitors of the trusted URI to an untrusted website.

7.2 Vulnerability indexing

This section discusses the identified and exploitable vulnerabilities during this research and presents a rating for each. The OWASP risk assessment method [21] served as a framework to classify and score the vulnerabilities. Due to the nature of this report we are primarily interested in the technical impact of the vulnerabilities. As such, the business impact analysis of the assessment methodology has been omitted.

7.2.1 User registration input may cause DoS for admin panel

For this vulnerability, refer to Table 2 and Appendix A. Due to a bug in the administrator’s view for listing registered users, the table containing the users may be displayed empty if a user’s email address contains XSS-like input. Although user registration is disabled by default, deployments with registration enabled may suffer from this bug since user administration in the usual fashion is not possible if the database contains such a user. The bug may be resolved by performing more extensive input sanitation for the registration form. Additionally, it would be advisable to reassess the Django HTML view itself, since invalid user input should under no circumstance affect other users of the platform.

Details with regard to this vulnerability are to be found in appendix A. Naturally, the risk of this vulnerability is highly influenced by whether the user registration panel is publicly accessible via the Internet. It should be noted that the vulnerability factors rating is inflated, as user registration is disabled by default. Additionally, the exploit may potentially be traceable to an IP address.

7.2.2 Insufficient parameter checking

For this vulnerability, refer to Table 3 and Appendix B. Due to insufficient access control validation, users with an activated account on a Seafile installation are able to share arbitrary libraries. Also, they are able to use another user’s library for creating a Wiki without the consent of that library’s owner.

Sharing libraries

As can be seen in appendix B, a malicious user with the possession of another user’s library ID can easily request a share URL, enabling anonymous visitors of this URL to download an entire library. As this library ID is included in the URL of any users web interface by default, copying and pasting a link to the library would be enough to reveal the ID.

If such a library is encrypted, metadata like file names are still readable in plain-text, and files, although encrypted, may be downloaded as well. The owner of the library will not be notified whenever this vulnerability is exploited. Also, as depicted in the appendix, he is not able to see his library is shared. The user abusing this flaw, will see the library to be appended to its list of shared libraries. In order to fix this vulnerability, the /share/ajax/get-download-link/ should perform sufficient permission checking. Only the owner of the library should be able to publicly share it. Due to the nature of this vulnerability, a basic level of access to the platform is required.

The vulnerability factors rating of this vulnerability are slightly skewed due to the fact that this exploit requires an activated user account on the platform. With regards to the technical impact, this exploit is non-disruptive. Data is only accessed and no risk of corruption exists. However, potentially all data can be disclosed which is detrimental for the confidentiality. Also, it is possible to trace this exploit back to the attacker as a user account is required on the platform.

Create arbitrary files

A similar flaw in the Seafile web API can lead to the creation of arbitrary *.md files in another user’s library. Due to not performing the ownership check on a library when enabling the Wiki for a user, any user is potentially able to pollute any other user’s library. Once again, as these files only contain text, no real technical impact can be identified.

7.2.3 Brute forcing password of shared library

For this vulnerability, refer to Table 4. The password field for opening a password-protected shared library is not protected with any kind of rate-limiting. Since the URI for opening a shared library is accessible to anonymous Internet users, and no password policy is being enforced for this field, this vulnerability is easily exploitable by a large group of users. When successful, a complete library may become available to anonymous users without the consent of the repository’s owner. Furthermore, besides the general HTTP access logs, no explicit logging of access attempts are being performed, leav-
### User registration input may cause DoS for admin panel

**Description:** Leveraging arbitrary Cross Site Scripting (XSS) code in the user registration panel causes the administrator panel to fail rendering.

<table>
<thead>
<tr>
<th>Threat Agent Factors</th>
<th>Rating</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Skill level</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Opportunity</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Motive</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Size</strong></td>
<td></td>
</tr>
<tr>
<td>Some technical skills</td>
<td>3</td>
</tr>
<tr>
<td>No access or resources required</td>
<td>9</td>
</tr>
<tr>
<td>Low or no reward</td>
<td>1</td>
</tr>
<tr>
<td>Anonymous Internet users (Depending on web panel access rights)</td>
<td>9</td>
</tr>
<tr>
<td><strong>Overall rating:</strong></td>
<td><strong>5.5</strong></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Vulnerability Factors</th>
<th>Rating</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Ease of discovery</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Ease of exploit</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Awareness</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Intrusion detection</strong></td>
<td></td>
</tr>
<tr>
<td>Automated tools available</td>
<td>9</td>
</tr>
<tr>
<td>Automated tools available</td>
<td>9</td>
</tr>
<tr>
<td>Obvious</td>
<td>6</td>
</tr>
<tr>
<td>Logged without review</td>
<td>8</td>
</tr>
<tr>
<td><strong>Overall rating:</strong></td>
<td><strong>8</strong></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Technical Impact</th>
<th>Rating</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Loss of confidentiality</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Loss of integrity</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Loss of availability</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Loss of accountability</strong></td>
<td></td>
</tr>
<tr>
<td>Minimal non-sensitive data disclosed</td>
<td>2</td>
</tr>
<tr>
<td>Minimal slightly corrupt data</td>
<td>1</td>
</tr>
<tr>
<td>Extensive secondary services interrupted</td>
<td>5</td>
</tr>
<tr>
<td>Possibly traceable</td>
<td>7</td>
</tr>
<tr>
<td><strong>Overall rating:</strong></td>
<td><strong>3.75</strong></td>
</tr>
</tbody>
</table>

**Table 2:** Risk evaluation: User registration input may cause DoS for admin panel

### Unprivileged public sharing of arbitrary user library

**Description:** Insufficient authorization checking when publicly sharing libraries allows for unauthorized access to files.

<table>
<thead>
<tr>
<th>Threat Agent Factors</th>
<th>Rating</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Skill level</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Opportunity</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Motive</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Size</strong></td>
<td></td>
</tr>
<tr>
<td>Network and programming skills</td>
<td>6</td>
</tr>
<tr>
<td>Some access or resources required</td>
<td>7</td>
</tr>
<tr>
<td>High reward</td>
<td>9</td>
</tr>
<tr>
<td>Authenticated users</td>
<td>6</td>
</tr>
<tr>
<td><strong>Overall rating:</strong></td>
<td><strong>7</strong></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Vulnerability Factors</th>
<th>Rating</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Ease of discovery</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Ease of exploit</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Awareness</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Intrusion detection</strong></td>
<td></td>
</tr>
<tr>
<td>Easy</td>
<td>7</td>
</tr>
<tr>
<td>Difficult</td>
<td>3</td>
</tr>
<tr>
<td>Unknown</td>
<td>6</td>
</tr>
<tr>
<td>Not logged</td>
<td>9</td>
</tr>
<tr>
<td><strong>Overall rating:</strong></td>
<td><strong>6.25</strong></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Technical Impact</th>
<th>Rating</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Loss of confidentiality</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Loss of integrity</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Loss of availability</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Loss of accountability</strong></td>
<td></td>
</tr>
<tr>
<td>All data disclosed</td>
<td>9</td>
</tr>
<tr>
<td>Minimal seriously corrupt data</td>
<td>1</td>
</tr>
<tr>
<td>Minimal secondary services interrupted</td>
<td>1</td>
</tr>
<tr>
<td>Possibly traceable</td>
<td>7</td>
</tr>
<tr>
<td><strong>Overall rating:</strong></td>
<td><strong>4.5</strong></td>
</tr>
</tbody>
</table>

**Table 3:** Risk evaluation: Unprivileged public sharing of arbitrary user library
Brute forcing password of shared library

Description: Shared, password protected libraries are susceptible to an online brute-force attack due to the lack of rate limiting on the credential verification.

<table>
<thead>
<tr>
<th>Threat Agent Factors</th>
<th>Rating</th>
</tr>
</thead>
<tbody>
<tr>
<td>Skill level</td>
<td>Some technical skills</td>
</tr>
<tr>
<td>Opportunity</td>
<td>No access or resources required</td>
</tr>
<tr>
<td>Motive</td>
<td>Possible reward</td>
</tr>
<tr>
<td>Size</td>
<td>Anonymous Internet users</td>
</tr>
<tr>
<td>Overall rating:</td>
<td>6.5</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Vulnerability Factors</th>
<th>Rating</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ease of discovery</td>
<td>Easy</td>
</tr>
<tr>
<td>Ease of exploit</td>
<td>Automated tools available</td>
</tr>
<tr>
<td>Awareness</td>
<td>Unknown</td>
</tr>
<tr>
<td>Intrusion detection</td>
<td>Logged without review</td>
</tr>
<tr>
<td>Overall rating:</td>
<td>5.75</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Technical Impact</th>
<th>Rating</th>
</tr>
</thead>
<tbody>
<tr>
<td>Loss of confidentiality</td>
<td>All data disclosed</td>
</tr>
<tr>
<td>Loss of integrity</td>
<td>Minimal slightly corrupt data</td>
</tr>
<tr>
<td>Loss of availability</td>
<td>Minimal primary services interrupted</td>
</tr>
<tr>
<td>Loss of accountability</td>
<td>Possibly traceable</td>
</tr>
<tr>
<td>Overall rating:</td>
<td>5.5</td>
</tr>
</tbody>
</table>

Table 4: Risk evaluation: Brute forcing password of shared library

ing the data owner unnoticed. As rate limiting is already being performed on the login page, it’s advised to enable it as well for opening a password-protected shared library.

Once again, the effective risk of this vulnerability is highly influenced by whether the user registration panel is publicly accessible via the Internet. The risk evaluation assumes that Internet access is possible. Therefore, the overall ratings may be inflated. With regards to the technical impact, a slight risk of data corruption is present as brute forcing the library requires a large amount of requests to the web server. As such, a slight risk of a DoS causing data corruption is present. Additionally, the exploit may potentially be traceable to an IP address.

8 Discussion

Judging by the reported security issues in the past, the initial hypothesis when going into the security audit was that the security of Seafiles platform as a whole would be below par. Surprisingly enough however, the contrary was true. A large series of the previously reported vulnerabilities in the product have been patched, are deemed less vulnerable than initially thought or are not applicable anymore. Additionally, the developers implemented a plethora of general security measures like captchas, periodic renewal of administrative credentials and rights management. Also the server-side configuration of Seafiles seems well thought out with correct file permissions, strong certificates and just enough communication between components. Of course, this is assuming that the secure install script is used. For a large part the security of Seafiles web front-end is provided by the tried and tested Django framework which allows the developers to leverage commonly used cryptographic libraries and built-in security measures such as Cross Site Request Forgery protection and SQL injection protection.

However, as presented in the results, not all of the reported issues were rectified which is worrying. Two of the GitHub issues regarding the reuse of Initialization Vectors and the possibility of retrieving metadata from an encrypted library are still present after nearly three years. It isn’t unreasonable to expect that these issues would be resolved by now. Additionally, reviewing the source code reveals that at points sloppy coding creates (significant) vulnerabilities. For example the brute forcing vulnerability of the password field. While the rest of the web application performs correct rate limiting, a single field was missed. Lastly, the fact that the included installer in the downloaded package by default is not a secure version while a secure version is available hurts the security of the average Seafile deployment as a whole.
Common vulnerabilities
In the preliminary research we evaluated common vulnerabilities for other private cloud storage solutions. Due to its popularity we were only able to find a significant amount of reported vulnerabilities for ownCloud. Naturally, as both platforms provide a web interface, the identified common vulnerabilities also apply to some extent to Seafile’s platform. All areas of interest have been covered in detail in section 7. During the audit we saw that the libraries used by Seafile did not have any reported vulnerabilities in this instance. However, this raises questions about whether these libraries are really secure or whether they are simply not thoroughly checked. ownCloud for example uses a different set of libraries which have been examined multiple times in the past by various independent reviewers. Also, incorrectly applying cryptography has also been found in Seafile as the issue with the reuse of Initialization vectors is still present. Lastly, incorrect parameter checking in the web client also allows for performing the brute force attack.

9 Conclusion
In this paper, we have evaluated the overall security of Seafile’s Private Cloud storage platform by performing a scoped security audit. More specifically we assessed the security of the web front-end ‘Seahub’. The product has been evaluated by testing for its susceptibility to common vulnerabilities as described in the OWASP Top Ten. Additionally, unresolved security issues reported in the past and vulnerability trends examined at competitor ownCloud have been taken into account. We present a series of vulnerabilities ranging in severity. Finally, the vulnerabilities have been scored following the OWASP Risk Rating Methodology.

Our results indicate that the Seafile as a storage platform has matured significantly in recent years due to the implementation of tried and tested frameworks. Still, this report presents a series of security vulnerabilities caused primarily due to sloppy coding. Most notably a high risk vulnerability which allows for the unprivileged public sharing of any arbitrary user library. Due to the limited resources and access rights required this vulnerability has been classified as high risk. Furthermore, shared, password protected libraries have been found to be susceptible to online brute-force attacks due to the lack of rate limiting on the credential field. Less severe is a vulnerability that allows for disrupting the administrator’s workflow by performing a Cross Site Scripting attack. Lastly, some previously reported vulnerabilities in the platform are still present.

Moving forward we advise a full code base review for Seafile, as a large amount of the uncovered vulnerabilities can be fixed by consistently applying security policies in the code. Additionally we suggest shrinking the overall code base, as the current redundancy increases the potential for error.

Considering the time frame and limitations imposed by the scope of this audit, we conclude that Seafile’s private cloud platform is susceptible to various attacks. These flaws can largely be attributed to invalid implementations of the framework and general coding mistakes.

10 Future Work
This report primarily focuses on the server components and modules of Seafile’s private cloud solution. Seafile also provides a plethora of methods to interface with the cloud platform from an end user perspective. At the time of writing there are official client applications available for the major mobile operating systems Android and iOS. Additionally desktop clients are available for Windows, Mac and Linux. A quick look through the issue tracker of these projects reveals that there is still work to be done regarding security.

Furthermore this audit only evaluated the security of Seafile’s free and open source ‘Community Edition’. However, as previously mentioned a Pro Edition of the platform is available which includes additional security sensitive features like Role Based Account Management, fine-grained folder permissions and authentication to Active Directory (AD) servers. Additionally, this version includes support for custom storage back-ends such as Ceph or Amazon Web Services (AWS) S3 which may be susceptible to security vulnerabilities themselves. Due to the overall impopularity of Seafile as a whole it may be interesting to perform a similar security audit on its commercial variant. The methodology used in this report could potentially be taken as a starting point.

Also, the strength of Seafile’s crypto implementation has been debated in the past. As this document aims to present a broad audit of the platform only a brief look at the randomness generator has been taken. A detailed investigation into these claims is required to rule out this potential vulnerability. Therefore we suggest a full code review of the back-end services. These are mainly written in C and may be susceptible to buffer overflows.

At last, it may be rewarding to further look into the vulnerability concerning the administrator panel Denial of Service. Although we were unable to find proof of possible code execution, there might be potential to do so.
References


Appendix A  – Register User

Figure A5: Register a new user

Figure A6: After registering, administrator lists all users
Appendix B – Share Library

$ curl --cookie "csrftoken=DyJ6haoOjmNNxMmZxvjhmhpYQHoI1J5d; sessionid=od3coy4dn2sg3c1hnnp3j65xi7fvznt1" -e https://seafile.testing.local/ -k https://seafile.testing.local/share/ajax/get-download-link/ -H "X-CSRFToken: DyJ6haoOjmNNxMmZxvjhmhpYQHoI1J5d" -H "X-Requested-With: XMLHttpRequest" -X POST --data "use_passwd=0&repo_id=a34c48dd-4e9f-4a32-98c0-1e54ff502697&p=%2F&type=d" {"token": "2364921741", "download_link": "https://seafile.testing.local/d/2364921741/"}

Listing 1: Share admin his repository (200library) by exploiting the /share/ajax/get-download-link/ URI

![Seafile interface](image1.png)

Figure B7: The owner of the library (admin) cannot see his library (200library) is shared to a public URI by viewing his libraries
Figure B8: The owner of the library (admin) cannot see his library (200library) is shared to a public URI by viewing his shared libraries.

Figure B9: The user who exploited the bug (test) can see he shared admin his library.