Fine penetration tests for fine websites

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

Pentest-Report Peergos Crypto & Software 05.2019

Cure53, Dr.-Ing. M. Heiderich, MSc. N. Krein, Prof. N. Kobeissi, MSc. D. Weißer

Index

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

Introduction

“Peergos is designed to give you back control over your data. It is an open source, secure, self hostable file storage and sharing platform. All your files are encrypted locally and your private keys never leave your machine.”

From https://peergos.org

This report documents the findings and impressions coming from a large-scale security assessment of the Peergos complex. Carried out by Cure53 in May and June 2019, this project entailed a dedicated penetration test, a source code audit and a design review of the Peergos file storage and sharing platform.

For context, it should be noted that Peergos is an open source project available on GitHub. Its main claim is about making use of end-to-end-encryption to offer private and secure file hosting. The software is managed by the Peergos team which commissioned Cure53 to conduct this review. Importantly, the tests and audits were funded by Protocol Labs, Inc.

In terms of resources, the work committed by Cure53 entailed a total of twelve person-days. Four members of the Cure53 were involved in the investigations, which were split into two distinct Work Packages (WPs). Specifically, one WP focused on the review of cryptography and design, while the other encompassed a penetration test and a source code audit targeting the Peergos’ web interface, server-side code, as well as its setup.

The project was rooted in a white-box method and Cure53 could access all information deemed viable as regards good coverage. Local builds were generally used for testing but Peergos also made an instance on a demo server available to Cure53 for the purpose of the assignment. It should be underlined that the design reviews were supported by extensive, largely public documentation.

The project progressed in a timely fashion. All communications during the investigations were done in a dedicated Slack channel into which Cure53 invited the Peergos team. The channel was used for asking pressing or emerging questions. Moreover, Cure53 sent over status updates on a regular basis and a live-reporting was requested and realized. This means that the Peergos team could immediately start tackling the fixes, especially for the more urgent problems. For example, this was the case with the XSS bug discussed in PGS-01-001. The exchanges were fluent and productive overall.

Moving on to the findings, this Cure53 project led to a discovery of ten issues. While the majority of seven problems could be linked to the review of the Peergos architecture, the remaining three were spotted through the penetration testing approaches. The numbers

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

were also split in relation to the problem type and here seven issues were evaluated as actual vulnerabilities and three items can be seen as general weaknesses. It is useful to note that only one finding - namely PGS-01-009 - reached a severity rating of "*High".*

In the following sections, this report will first shed light on the scope by supplying technical details about the Peergos scope examined during this project. The document then furnishes case-by-case descriptions of the findings spotted over the two phases of this project, featuring both technical details and possible mitigation going forward when applicable. Notes on fixes are also incorporated into the tickets when applicable. Based on the results of this summer 2019 assessment, Cure53 issues a broader verdict about the privacy and security posture of the tested items. Conclusions are supplied in the final section of this document and pertain to the tested Peergos file hosting and sharing tool, as well as its surroundings formed by the server, architecture and design.

Scope

- **Peergos Crypto Design & Software Implementation**
  - **WP1: Crypto Design, Crypto Libraries and general Security Architecture**
    - General concepts are available at https://book.peergos.org/
    - Additional design and architecture documentation was shared with Cure53
  - **WP2: Web Interface, underlying server & code**
    - Sources are publicly available
      - https://github.com/peergos/peergos
      - https://github.com/peergos/web-ui
    - A demo server was made available for Cure53
      - https://demo.peergos.net

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

Identified Vulnerabilities

The following sections list both vulnerabilities and implementation issues spotted during the testing period. Note that findings are listed in chronological order rather than by their degree of severity and impact. The aforementioned severity rank is simply given in brackets following the title heading for each vulnerability. Each vulnerability is additionally given a unique identifier (e.g. *PGS-01-001*) for the purpose of facilitating any future follow-up correspondence.

PGS-01-001 Web: Response-splitting leads to reflected XSS *(Medium)*

It was discovered that the Peergos web application is vulnerable to reflected XSS via response-splitting. Since every instance of the page (*tab/window*) has its own state, it is not possible to exploit the issue and, for example, steal cookies or navigate the site on the victim's behalf. In a real-world scenario, the attacker would send a crafted XSS payload with an evil login page to the victim. The evil page would need to display a message like "*Login to display file"*. The multistage nature makes successful exploitation more difficult and explains the corresponding "*Medium"* ranking. A Proof-of-Concept is given by the following URL.

PoC:
https://demo.peergos.net/peergos/v0/core/meow%0d%0a%0d%0a%3cb%3ea%3csvg%20onload%3dalert%281%29%3e/

The problem is caused by the lack of encoding when it comes to user-input reflected via an HTTP header. This is shown in the following code snippet.

Affected File:
peergos/server/corenode/HttpCoreNodeServer.java

Affected Code:

```
    switch (method)
    {
[...]
            break;
        default:
            throw new IOException("Unknown method "+ method);
    }

    dout.flush();
    dout.close();
    byte[] b = bout.toByteArray();
    exchange.sendResponseHeaders(200, b.length);
    exchange.getResponseBody().write(b);
```

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

```
} catch (Exception e) {
    Throwable cause = e.getCause();
    if (cause != null)
        exchange.getResponseHeaders().set("Trailer", cause.getMessage());
    else
        exchange.getResponseHeaders().set("Trailer", e.getMessage());
```

It is recommended to properly encode all user input that is reflected to the page. In this particular scenario URL-encoding is needed. The issue was reported during the testing phase and has been fixed immediately, the fix was verified by Cure53.

Note: *The fix deployed by Peergos was successfully verified during the testing phase.*

PGS-01-003 Web: Missing *Content-Type* in error messages leads to XSS *(Medium)*

It was possible to find another reflected XSS vulnerability that is slightly different from the one demonstrated in PGS-01-001. In the application's *MutationHandler*, response-splitting is not required because the error message contains reflected *path*-parameters of the URL on its own. Since these are user-controlled, this creates a sink that can be directly exploited. In browsers such as Firefox this can be demonstrated with the PoC provided next.

Payload.html:

```
<a type="text/html"
href="https://demo.peergos.net/peergos/v0/mutable/test.html;<svg%20onload=
%22alert(1)%22>?
owner=z59vuwzfFDostraunJTvWqa1ptNBVR1vnBASbVAWMjdGmoNK9Uk9d7z&writer=z59vuwzfFDp
2WEJAMaS66uokSf5MNWYAVuNrVJ266oErZS8viSi3oVU">CLICKME
```

Response:

```
HTTP/1.1 400 Bad Request
Date: Mon, 03 Jun 2019 09:06:24 GMT
Trailer: Unknown+method+test.html%3B%3Csvg+onload%3D%22alert%281%29%22%3E
Strict-transport-security: max-age=31536000
Content-Length: 48

Unknown method test.html;<svg onload="alert(1)">
```

As one can see from the response, upon clicking on the *CLICKME* of *Payload.html,* Firefox will automatically assume a *Content-Type* of *text/html* and thus happily runs the embedded JavaScript code.

Affected File:
peergos/src/peergos/server/net/MutationHandler.java

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

Affected Code:

```
} catch (Exception e) {
    LOG.log(Level.WARNING, e.getMessage(), e);
    exchange.getResponseHeaders().set("Trailer", e.getMessage());
    exchange.sendResponseHeaders(400, 0);
    OutputStream body = exchange.getResponseBody();
    body.write(e.getMessage().getBytes());
} finally {
```

It is recommended to strictly force a *Content-Type* of *text/plain* when sending error responses that should not be automatically rendered as HTML. Additional output-validation in a form of HTML escaping should also be seen as optional in this realm.

Note: *The fix deployed by Peergos was successfully verified during the testing phase.*

PGS-01-004 Crypto: Password hashing salt insufficiently unique *(Low)*

Peergos relies on *Scrypt*, which is a state-of-the-art password hashing function. *Scrypt* is used for hashing user-passphrases and for subsequently generating key material. However, it was determined that the salt used for Peergos password hashing was not sufficiently unique. This is because it is composed purely of the username connected with the user.

Affected Files:

- *src/peergos/shared/crypto/password/PasswordProtected.java*
- *src/peergos/shared/user/ScryptGenerator.java*
- *src/peergos/shared/user/SecretGenerationAlgorithm.java*

Affected Code:

```
public static SecretGenerationAlgorithm getDefault() {
    return new ScryptGenerator(ScryptGenerator.MIN_MEMORY_COST, 8, 1, 32);
}

[...]

return new ScryptGenerator(memoryCost, cpuCost, parallelsm, outputBytes);

[...]

static SecretGenerationAlgorithm getDefault() {
       return new ScryptGenerator(ScryptGenerator.MIN_MEMORY_COST, 8, 1, 96);
}
```

It is recommended for Peergos to extend the salt beyond username and add either a fixed string or a randomized nonce to it.

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

Note: *This issue was addressed in the* `0ef859d243ca79f2ebe2fdbbe74ba52666cf0689` *commit and the changes were verified by Cure53.*

PGS-01-007 Crypto: Switching from *SECIO* to *TLS 1.3* in *IPFS* (*Medium*)

It was observed that Peergos uses *IPFS* with *SECIO* configured as means for establishing the transport layer protocol. Using *SECIO* is considered to be quite risky for several reasons listed next.

1. *SECIO* is not formally specified anywhere; the SECIO implementations appear to be ad-hoc.
2. No protocol security guarantees are specified or formally attributed to *SECIO.*
3. Critical-severity vulnerabilities have recently appeared in *SECIO* implementations, including the *Rust* implementation of *libp2p*[1]. The latter is important because Peergos uses the Go implementation of the same library.

It is recommended to instead use *IPFS* with *TLS 1.3* for transport layer encryption. This is due to the fact that the *TLS 1.3* is a formally specified and formally verified protocol with stringent implementation standards.

Note: *This issue was addressed in* `1d042fa24e41f256d66428ed0072b4e1c377f0de` *commit and the changes were verified by Cure53.*

PGS-01-008 Crypto: Username allowed as password (*Medium*)

Peergos implements strength checks on user-passphrase input. Namely, it checks for short passwords as well as passwords that are known to be commonly employed. However, it was observed that Peergos makes no restrictions for users who input their own username as their password. Especially with the high cryptographic value of the passwords in Peergos (i.e. in terms of all of user-keys being directly derived from their password), this constitutes a significant security risk.

It is recommended to forbid the practice of having passwords equal to the username. Optionally, it can even be disallowed for the username to appearing anywhere within the password/

Note: *This issue was addressed in* `41d2232b1be474bc68900da52adeed01d979c0e5` *commit and the changes were verified by Cure53.*

[1] https://rustsec.org/advisories/RUSTSEC-2019-0004.html

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

PGS-01-009 Crypto: No restriction or warnings on weak password changes *(High)*

Peergos implements stringent checks on user-passwords during signup. However, it was discovered that no checks whatsoever were implemented if a user were to decide to change their password *after* signup. This would allow the user to set their password to values as simple as *"1"* or *"abc"* without any warnings issued by Peergos.

Since all cryptographic key material for a Peergos user is derived from their passphrase, a weak password could have catastrophic consequences. It could foster quick wordlist-based dictionary attacks, even when a strong password hashing function, such as *Scrypt*, is used with high-cost parameters.

It is strongly recommended to implement the same checks for password strength as those in place for the initial account registration.

Note: *This issue was addressed in* `3880ab43600de5b3c51d850cf9057f224465dbea` *commit and the changes were verified by Cure53.*

PGS-01-010 Crypto: Unclear difference on mutual authentication levels *(Medium)*

The Peergos "*Book*" mentions three different threat models[2] intended to represent and consider different types of users - a casual one, a slightly paranoid one and a more paranoid one. Their characteristics are supplied next on the basis of the documentation provided.

*"**Casual user:***

- *Trusts the SSL certificate hierarchy and the domain name system.*
- *Is happy to run Javascript in their browser.*
- *Trusts TLS and their browser (and OS and CPU ;-))*

Such a user can interact with peergos [sic] *purely through a public web server that they trust over TLS.*

Slightly paranoid user:

- *Doesn't trust DNS or SSL certificates*
- *Is happy to run Javascript served from localhost in their browser*

This class of user can download and run the Peergos application and access the web interface through their browser over localhost.

[2] https://book.peergos.org/security/threats.html

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

More paranoid user:

- *Doesn't not [sic] trust the SSL certificate system*
- *Doesn't trust DNS*
- *Doesn't trust javascript*

This class of user can download the Peergos application (or otherwise obtain a signed copy), or build it from source. They can then run Peergos locally and use the native user interface, either the comand [sic] *line or a FUSE mount. Once they have obtained or built a copy they trust, then they need trust only the integrity of TweetNacl cryptography (or our post-quantum upgrade) and the Tor architecture."*

The difference between these three threat models is stark. It is completely impossible for the "*casual user*" to obtain any mutual authentication guarantees whatsoever, given that they are running code from a potentially malicious server.

In spite of this, no effort is made in the Peergos web application to either allow users to indicate which level of security they are expecting (and for Peergos to offer the appropriate option as a response) or even for Peergos to illustrate to users that such a difference in security exists.

Other high-security web applications take such precautions in order to inform users about similarly potent threat model differences. For example, *MyEtherWallet*[3] forces all users to complete a mandatory tutorial before using the service. This appeared to be a warranted precaution, since a DNS server compromise in 2018 led to exactly the sort of attack that would compromise the "*casual"* Peergos users[4].

It is recommended for Peergos to adopt similar measures to either inform users on the potentiality of various risks that apply to different users. Alternatively, the users could be asked to describe their threat model and subsequently provided with feedback on how to best access Peergos in a way that is appropriately adapted to their security needs and expectations.

[3] https://www.myetherwallet.com/
[4] https://news.bitcoin.com/myetherwallet-servers-are-hijacked-in-dns-attack/

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

Miscellaneous Issues

This section covers those noteworthy findings that did not lead to an exploit but might aid an attacker in achieving their malicious goals in the future. Most of these results are vulnerable code snippets that did not provide an easy way to be called. Conclusively, while a vulnerability is present, an exploit might not always be possible.

PGS-01-002 Web: Bogus usernames not prevented by server-side checks *(Info)*

It was discovered that the sanity check for usernames happens solely on the client-side, thus allowing registration of bogus usernames. This can be achieved by executing the procedure outlined next.

Steps to reproduce:

- Open the *signup* page in Chrome.
- Open debugging console.
- Set a breakpoint on the following line:
- `if(!usernameRegEx.test(that.username)) {`
- Enter the desired username and submit.
- Enter into the JS console: `usernameRegEx=/./`
- Continue the JS process.

Besides creating non-usable accounts, no impact was discovered and the issue's potential was ranked as "*Informational*" However, in order to maintain consistency, it is recommended to implement a server-side check as well.

Note: *This issue was addressed in the* `927d1ae267889dd894aefb1bace24ea14d6e6983` *commit and the changes were verified by Cure53.*

PGS-01-005 Crypto: Hashing in *Merkle-CHAMP* PKI recommended *(Info)*

Peergos uses a lookup table for its PKI-based username-to-public-key lookups. This structure is then distributed on a "Merkle-CHAMP" structure in order to obtain guarantees over the integrity and authentication. These are supposed to be similar to those commonly seen in blockchain data structures. In this lookup table, usernames are currently stored directly as keys and then bound to the user's public key as a value.

Peergos is encouraged to move away from this approach. Instead, it is recommended to store the hash of the username. This has two minor benefits:

- The size of the PKI structure grows linearly with the number of users since both key and value fields would become of a known fixed length.

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

- Exposing the PKI structure to a random observer does not necessarily reveal the list of usernames on a particular Peergos network.

This recommendation is completely optional and the benefits it provides to the Peergos PKI structure are minimal. However, the cost of such implementation is also minimal.

Note: *This issue was addressed in the* `001a082af730556c891eb54683033220aa3047cd` *commit and the changes were verified by Cure53.*

PGS-01-006 Crypto: Recommendations for side-channel-resistant primitives *(Info)*

It was found that Peergos used implementations of public key cryptography primitives, namely *Ed25519* and *Curve25519,* in JavaScript and Java. While the implementations themselves appear to be correct and sound, both JavaScript and Java are languages interpreted on a high-level. In the current design, it is impossible to mitigate side-channel attacks - such as timing attacks or key recovery from memory - with a good level of certainty.

It is recommended to review new research that allows for side-channel-resistant primitives, even in JavaScript environments. One example is the proposal from *HACL-WASM*, which includes formally verified cryptographic libraries compiled to *WebAssembly*[5].

Ultimately, moving to these primitives will allow Peergos clients and servers to offer a higher level of assurance against functional incorrectness at the primitive level. Furthermore, it signifies better resistance to side-channel attacks on Peergos infrastructure or client devices.

[5] https://eprint.iacr.org/2019/542/20190522:085258

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

Conclusions

This summer 2019 security assessment of the Peergos complex concludes with quite convincing evidence of the project being on the right track from a security standpoint. After examining the scope for twelve days, four members of the Cure53 team can clarify certain motivations between this rather positive verdict. On the one hand, the number of ten issues on the Peergos items might seem quite extensive. On the other hand, the findings did not signify paramount. Moreover, the problems were easy to fix and - crucially - the efforts from the Peergos team demonstrated that they are being addressed correctly in-house.

Some areas were more prone to findings than others and these are worth-discussing in more detail. Regarding web security, it can be said that the platform turned out robust and managed to resist many attempts at typical compromises. Common web security flaws were mostly avoided, for example with no findings linked to file upload and validation attacks. To emphasize, no actual server-side issues like RCE or ACL bugs could be spotted. However, the platform suffered from XSS, mostly due to handling the HTTP responses poorly. On one occasion, user-controlled input allowed a vulnerability called *“HTTP response-splitting”.* This flaw essentially made it possible for attackers to spoof arbitrary HTTP responses and led to XSS addressed in PGS-01-001. Another example was found in PGS-01-003 where generated error messages were directly reflected back with user-controlled input. On the plus side, both tickets were very quickly addressed and verified by Cure53 again. Across all other aspects, the platform was said to be in a good shape.

In terms of cryptography, it needs to be stated that Peergos consistently relies on known-secure, well-studied primitives and protocols. Despite being a large-scale project with a complex design, Peergos manages to meet the security goals related to crypto. The cryptographic implementation was well-documented, fostering a comprehensive level of depth being reached during the tests. Some minor issues were found as regards passphrase security (see PGS-01-004 and PGS-01-008); again, these issues were confirmed as fixed. Another noteworthy flaw was found in the core infrastructure element of *IPFS*, which was configured to use the insecure *SECIO* protocol instead of the well-studied *TLS 1.3* protocol (see PGS-01-007). There were two more severe issues that should be elaborated on. The first on, referred under PGS-01-009, made it possible for users to set extremely weak passphrases. From there, keys for their entire account could be obtained trivially with brute-force. Conversely, PGS-0-010, discusses how Peergos users are not being made aware of the differences between the theory and the real-world security that can vary depending on how the Peergos application is accessed. This managed to happen despite these differences being mentioned in the threat model offered by the technical document of Peergos.

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

To conclude, the security of the Peergos web application complex should be seen as passing this evaluation. This is supported by excellent customer communication and coordination. It is important to highlight that various issues that have been reported during the testing phase were addressed immediately. The back and forth discussions with the Peergos team in-house were very fruitful. Drawing on the initial handling of the fixes, Cure53 feels confident that the remaining problems will be rectified in no time. Once all issues are addressed, Peergos can ship a product that is clearly safe to use. The Peergos clearly aims at offering a product that sets the high security standard for its application security premise.

Cure53 would like to thank Ian Preston from the Peergos team for his excellent project coordination, support and assistance, both before and during this assignment. Special gratitude needs to be extended to Protocol Labs, Inc. for sponsoring this project.