Fine penetration tests for fine websites

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

Pentest-Report Frame v0.2 07.2019

Cure53, Dr.-Ing. M. Heiderich, Prof. N. Kobeissi, M. Kinugawa,
BSc. T.-C. “Filedescriptor” Hong, N. Hippert

Index

Introduction

“Frame is an OS-level Ethereum interface that lets you use standalone signers, such as a Ledger or Trezor, to interact with dapps and the Ethereum network”

From https://frame.sh/

This report documents the findings of a security assessment targeting the 0.2 branch of the Frame Ethereum interface software. Carried out by Cure53 in July 2019, this project entailed both a penetration test and a source code audit. Only four security-relevant issues were spotted.

For context, it should be noted that Cure53 has previously examined the software in scope back in August 2019. Between then and July 2019, several new features have been added and these - besides the regular scope - were treated with extra scrutiny. As for the resources, Cure53 approached the Frame 0.2 branch scope with a team of five testers. The investigations of the running software and its code required a budget of eight days, though some days have been preserved for the sake of another Cure53 assessment of Frame planned for late 2019.

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

In terms of specific tasks, the scope was split into three Work Packages. The software on its own - as a whole and across components - was subjected to Cure53 analysis in WP1, which further scrutinized the integration of various signers. In WP2, Cure53 examined the locally spawned servers, while the handling of third-party binary download and execution constituted the key aspects of WP3.

The project started on schedule and progressed in an efficient manner. The communications during the tests were done in a dedicated channel on the Gitter chat platform. Over the course of the assessment, Cure53 reported on the status and asked questions. In this channel, Cure53 also received information about the code changes that were made since the last analysis, so that the already reported issues could be reexamined and the new issues could be resolved promptly. Thus, the testing team was able to review fixes on the fly while the test was still ongoing. All in all, the test went fluently and without any noteworthy problems.

Three findings were marked as vulnerabilities with variable severities and one was noted as a general weakness with low exploitation potential. Notably, only a single discovery received a severity ranking of "*High"* while other flaws signified much less pronounced risks. Foreshadowing the conclusion, a temporal look shows that the Frame project progressed considerably in terms of security. The number of issues - when compared to August 2018 - has decreased for this July 2019 assessment.

In the following sections, the report will first present the scope by listing all WPs. It then moves on to tickets which shed light on the discoveries, doing so in a case-by-case manner. Alongside technical aspects like PoCs, Cure53 furnishes mitigation advice when applicable. The report closes with a conclusion in which Cure53 summarizes the project and issues a verdict about the tested scope. Conclusions about the security and privacy posture of the Frame 0.2 branch software interface are supplied in the final section of this document

Scope

- **Frame 0.2 and its new features, see below**
  - **WP1:** Frame Electron Application that integrates and interfaces various *Signers*
  - **WP2:** Local Servers and Services spawned by the Frame Electron Application
  - **WP3:** Secure Handling of Third-Party Binary Download and Execution Features
- **Sources were made available to Cure53**
  - https://github.com/floating/frame/tree/0.2
- **Cure53 was able to create builds, these were run locally and tested against**

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. *FRM-02-001*) for the purpose of facilitating any future follow-up correspondence.

FRM-02-001 Frame: *Hot signer* accounts allow weak passphrase *(Low)*

Note: *This vulnerability was live-reported and immediately fixed by the Frame security team. The fix was verified by the Cure53 team.*

It was found that passphrase entry dialogs of the *hot singer* accounts accept weak passwords, such as *123456*. Given that keystores store long-term keys susceptible to offline attacks, stronger passwords are necessary to achieve an acceptable level of resilience.

It is recommended for Frame to begin to enforce the use of random passphrases and prioritize such a strategy over the keystore passwords. In the event that a user insists on a keystore password, it is recommended to impose a minimum length of 12 or 14 characters. Furthermore, a password strength measurement library, for instance *zxcvbn*[1], can be used to provide more nuanced metrics regarding the passwords' strength.

FRM-02-002 Frame: User-password utilized directly as encryption key *(High)*

Note: *This vulnerability was live-reported and immediately fixed by the Frame security team. The fix was verified by the Cure53 team.*

It was found that the user's password is utilized directly as the encryption key after being hashed only once using *SHA-256*. Coupled with the fact that weak passwords are allowed (as documented in FRM-02-001), this could result in cipher-text that is vulnerable to dictionary, brute-force and other similar attacks. Furthermore, this approach would lead to users with the same password having the same encryption key.

Affected files:
crypt/index.js
signers/hot/HotSigner/worker.js

[1] https://github.com/dropbox/zxcvbn

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

Affected code:

```
const stringToKey = pass => {
  const hash = crypto.createHash('sha256').update(pass)
  return Buffer.from(hash.digest('hex').substring(0, 32))
}

const encrypt = (string, pass, cb) => {
  try {
    const iv = crypto.randomBytes(16)
    const cipher = crypto.createCipheriv('aes-256-cbc', stringToKey(pass), iv)
    const encrypted = Buffer.concat([cipher.update(string), cipher.final()])
    cb(null, iv.toString('hex') + ':' + encrypted.toString('hex'))
  } catch (e) { cb(e) }
}

const decrypt = (string, pass, cb) => {
  try {
    const parts = string.split(':')
    const iv = Buffer.from(parts.shift(), 'hex')
    const decipher = crypto.createDecipheriv('aes-256-cbc', stringToKey(pass),
iv)
    const encryptedString = Buffer.from(parts.join(':'), 'hex')
    const decrypted = Buffer.concat([decipher.update(encryptedString),
decipher.final()])
    cb(null, decrypted.toString())
  } catch (e) { cb(e) }
}
```

It is recommended to use a salted, secure password hashing function, such as *Scrypt,* instead[2].

FRM-02-004 Frame: Several DoS issues via request to local server *(Low)*

It was discovered that the Frame desktop application's local server can be crashed when a crafted request is sent. As a result, the application can be disabled from visiting any website which sends a malformed request to *localhost.* Three examples of crashes are shown below. By executing the PoC's JavaScript from any origin, the Frame desktop application will crash.

PoC #1:

```
fetch('http://127.0.0.1:1248/',{"method":"post","body":'{"id":1,"method":
{"toString":null},"params":{"toString":null}}'})
```

[2] https://www.tarsnap.com/scrypt.html

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

Affected File #1:
https://github.com/floating/frame/blob/4cf7336d944b8bea6dd84ea49f454c1613eaa885/main/api/http.js#L42

Affected Code #1:
```
let payload = validPayload(Buffer.concat(body).toString())
if (!payload) return
log.info('req -> | http | ' + req.headers.origin + ' | ' + payload.method + ' |
-> | ' + payload.params)
```

Thrown Error #1:
```
TypeError: Cannot convert object to primitive value
```

PoC #2:
```
ws=new WebSocket('ws://127.0.0.1:1248/');
ws.onopen=function(){
  ws.send('{"id":1,"method":{"toString":null},"params":{"toString":null}}');
}
```

Affected File #2:
https://github.com/floating/frame/blob/4cf7336d944b8bea6dd84ea49f454c1613eaa885/main/api/ws.js#L38

Affected Code #2:
```
let payload = validPayload(data)
[...]
log.info('req -> | ' + (socket.isFrameExtension ? 'ext | ' : 'ws | ') + origin +
' | ' + payload.method + ' | -> | ' + payload.params)
```

Thrown Error #2:
```
TypeError: Cannot convert object to primitive value
```

In addition, the same issue as *FRM-01-004*, which was found in the last test, has been discovered during this test iteration again.

PoC #3:
```
fetch("http://127.0.0.1:1248/",
{"method":"post","body":'{"id":1,"method":"aaa"}'})
```

Thrown Error #3:
```
Uncaught NodeError: Unhandled error. ([object Object])
```

For items #1 and #2, it is recommended to check that all passed *JSON* properties have the expected data-types before having them accessed. This check can be done by using

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

the *typeof* operator, *Array.isArray()* or *Object.prototype.toString* method. As for the issue #3, it is recommended to ensure that all undefined commands are handled properly.

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.

FRM-02-003 Frame: Outdated and vulnerable Electron version *(Info)*

The Frame desktop application currently relies on Electron 4.0.8, which is an outdated version. In the following, it is shown that the *navigator.userAgent* property indeed returns Electron 4.0.8.

Value from *navigator.userAgent*:

```
Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/69.0.3497.128 Electron/4.0.8 Safari/537.36
```

As described in the Electron's release notes[3], some security issues in Chromium have been fixed with version 4.2.4. In order to protect users from the platform's vulnerabilities, it is recommended to always use the latest version of Electron. In addition, it should be noted that the latest Chromium is not used, even with the latest Electron. Cure53 advises the Frame team to consider the application's development in the context of Electron releases always coming later than their Chromium counterparts.

Conclusions

As already noted in the *Introduction,* the Frame 0.2. branch should be seen as more secure than it has been when Cure53 first examined it a year ago. In light of the findings accumulated during this July 2019 assessment, the Cure53 testing team can attest the good quality of the project and the generally positive impression gained about the robustness of the scope.

In terms of technical details, it can be stated that mostly minor issues were spotted. This holds especially for the key handling of the newly added *hot signer* feature, which was of particular interest for this security testing iterations. While other flaws include Denial-of-Service on the local server via specifically crafted requests, Cure53 feels obliged that code quality found on the 0.2. branch is not always up to par. As can be seen from FRM-

[3] https://github.com/electron/electron/releases/tag/v4.2.4

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

02-004, many of the discoveries could have been prevented if the Frame team relied on libraries rather than own writing of the network logic.

It should be noted that due to the dynamic nature of JavaScript, user-input needs to be sensitively handled to prevent runtime errors coming from specially crafted requests, as demonstrated in FRM-02-004. At the same time, the application handles almost all of its data in a safe manner and no critical exploits (e.g. XSS) have been found. Concurrently, FRM-02-004 proves that there is room for improvement as far as handling of user-data is concerned.

Not much can be stated in terms of cryptography as the deployment is quite minimal. Most of the handling has to do with encrypting sensitive data locally and Cure53 found that encryption primitives in use are adequate. Conversely, the encryption key derivation is weak and vulnerable to brute-force attacks (FRM-02-002). This is made worse by the fact that very weak passwords are allowed (FRM-02-001).

Moving on to Electron, Cure53 needs to comment that some platform-specific Electron problems have been found during the last test. This time, the only issue was attributed to an older version in use, as documented in FRM-02-003 Therefore, Cure53 is happy to report that platform-specific issues connected with Electron have been successfully addressed. Next, the fact that no noteworthy bugs stem from the update feature illustrates that the associated risks are well-understood and the ability to make the deployment reliant of mature and known components has been utilized. This enforces best practices and makes it hard for major flaws to be introduced.

In conclusion, Cure53 can ascertain that the Frame 0.2. Ethereum interface software branch makes a good impression from a security standpoint. The inspection carried out in July 2019 confirms that only some minor issues still affect the applications and libraries in scope. In other words, the development team is certainly on the right track when it comes to procuring a safe product.

Cure53 would like to thank Jordan Muir and Philip Prophet from the Frame team for their excellent project coordination, support and assistance, both before and during this assignment.