FYEO

Dynamic wallet testing report

Lace wallet

Presented by:

FYEO Inc.

PO Box 147044

Lakewood CO 80214

United States

Executive Summary

Overview

IOHK has engaged FYEO Inc. to perform dynamic testing of potential threats to the wallet implementation and to create a system that enables continuous security testing of future releases of the extension.

This report is a preliminary report presented during the assessment and presents the results of a dynamic security audit and tests performed on the wallet browser extension.

The purpose of this audit was to evaluate the security of the wallet and ensure that it does not leak any user secrets or expose the users of the wallet to any additional risks that could not be identified by the code audit. The scenarios tested in the audit and the automated tests implemented are the results of a threat modeling exercise performed together with the team.

The assessment was conducted remotely by the FYEO Security Team. Testing took place between December 2022 and February 2023, and focused on the following objectives:

- To provide the customer with an assessment of their overall security posture and any risks that were discovered within the environment during the engagement.
- To provide a professional opinion on the maturity, adequacy, and efficiency of the security measures that are in place.
- To identify potential issues and include improvement recommendations based on the results of our tests.

This report summarizes the engagement, tests performed, and findings. It also contains detailed descriptions of the discovered vulnerabilities, steps the FYEO Security Team took to identify and validate each issue, as well as any applicable recommendations for remediation.

Key Findings

The security audit and automated tests did not yield any high severity findings, and the wallet was found to be securely implemented with no significant vulnerabilities.

- The wallet does not leak any user secrets, and user data is protected through the use of strong encryption algorithms.
- Some potential weaknesses were detected in the way the wallet interacts with the web. E.g., web3 interface
- These medium risk vulnerabilities could lead to users of the wallets being the target of fraud and other indirect risks to wallet users
- The team has acknowledged these vulnerabilities and are working to remediate them in an upcoming release

Identified findings and recommendations

The following issues have been identified during the testing period. These should be prioritized for remediation to reduce the risk they pose. All of the identified potential vulnerabilities were related to the Dapp integration

Dapp integration

- Possible to loop trust of wallet through malicious website - **Medium**
- Possible to initiate wallet trust from non ssl encrypted sites - **Medium**
- Possible to initiate trust of wallet from private ip addresses - **Medium**
- Lack of blacklist for malicious sites - **Informational**

Detailed Findings and Tests Performed

Threat Modeling Methodology

The threat modeling exercise is a process by which potential threats, such as structural vulnerabilities or the absence of appropriate safeguards, can be identified, enumerated, and mitigations can be prioritized.

The purpose of this threat model analysis is to provide defenders (product owners) with a systematic analysis of what controls and defenses need to be included given the nature of the system, the probable attacker's profile, the most likely attack vectors, and the assets most desired by an attacker.

The source used to produce this document has been a workshop between the consultants from FYEO Inc. together with the stakeholders from IOHK.

The goals of this exercise were:

1. To get a better understanding of the functionality and the components of the system

2. To identify the most likely and most significant threats to the solution and prioritize them

3. To identify which threats could and should be verified via functional testing of the extension

During the threat modeling workshop, a simplified version of the industry-standard methodology of threat modeling as described by Microsoft was used.

Identified Threat Scenarios

Here we list the potential threat scenarios that were identified during the threat modeling and how the scenario was translated into tests.

Laptop stolen by malicious partner (requires physical access)

An attacker with physical access can recover decryption keys for hard drive encryption by dumping the RAM of a computer while the system is running and the hard drive is mounted. This technique is known as a "cold boot attack" and it can be used to extract sensitive information from the computer's memory, including encryption keys and other secrets.

Even if the laptop or end device was stolen or acquired by a threat actor it would still not be possible to extract the private key from the end device. This was proven by implementing tests to dump the physical memory of the machine. This simulated cold boot attacks and several other attack scenarios.

During the tests, no critical information, such as, private keys or mnemonics, were found in the analyzed ram of the target machine

Hostile site executing script via xss/csrf or compromised/hacked server

An attacker can execute code on a victim's browser through a variety of techniques, including cross-site scripting (XSS) and cross-site request forgery (CSRF) attacks, as well as, compromising a server or website.

Here's how each of these techniques can be used to execute code on a victim's browser:

1. Cross-site scripting (XSS): In an XSS attack, an attacker injects malicious code into a website that is then executed by a victim's browser. This can be done through vulnerabilities in the website's code, such as input validation errors or unsecured user input fields. Once the victim visits the compromised website, the attacker's code is executed on their browser, allowing the attacker to steal sensitive information or perform other malicious actions.
2. Compromised server or website: An attacker can also compromise a server or website and use it to deliver malicious code to a victim's browser. This can be done through a variety of techniques, such as exploiting vulnerabilities in the

server's code or using social engineering to trick a legitimate user into providing access to the server. Once the attacker has control over the server or website, they can deliver their malicious code to any visitors to the site, potentially executing code on the victim's browser.

In this case, potential vulnerabilities have been identified that make it possible for an attacker to initiate transactions as well as initiate trust from the compromised site.

HW wallet stolen

If the hardware wallet is stolen, that is a risk to the end user, however, the wallet is protected by pincode and the mnemonic of the hardware wallet would be needed to sync that hardware wallet to a new instance of the extension. These tests have been implemented and no vulnerabilities have been identified that make it possible to use a stolen hardware wallet without knowing the seed phrase.

Keyboard sniffing and attacks via Other extensions? (possible bad browser process sandboxing)

There has historically been a number of rogue chrome extensions that have been spying on other extensions and user activity. This is a likely scenario and therefore tests were implemented to simulate a user keyboard sniffing the tab context to simulate keyboard sniffing from another chrome extension. While the attack is still possible, it is only possible to sniff the keyboard strokes from the browser tab context and not from other chrome extensions. This makes this attack vector unlikely since the critical information, such as, password, mnemonic, etc. is inputted into the chrome popup directly.

Social engineering / phishing attacks

Social engineering attacks and phishing emails leading to malicious websites are a common way to defraud crypto and wallet users. As an example, in a similar domain attack, an attacker registers a domain name that is very similar to a legitimate website, but with a slight variation in spelling or domain extension. For example, they may register "gooogle.com" instead of the legitimate "google.com." The attacker then creates a fake page on the similar domain that looks identical to the legitimate site, and sends phishing emails or social media messages to victims, urging them to visit the fake page and enter their credentials and/or connect their wallet.

Here, some of the identified potential weaknesses in the Dapp integration makes these attacks possible and likely.

Copy Key while in the cache of Lace and Chrome

As an attacker with total control of the users machine, it is possible to access the RAM of the browser process and read the Random Access Memory of the browser to access the javascript heap where the unencrypted master key resided at the time of signing.

While this attack is indeed possible, it is a very unlikely attack and the extension protects the decrypted key and keeps it in RAM for as short of a period of time as possible implementing adequate security protection against these sorts of attacks. Our implemented tests show that these types of attacks are highly unlikely to succeed and would require a very advanced malware to implement continuous scanning of RAM.

Changing the destination address in the browser cache

Web cache poisoning is an attack that exploits vulnerabilities in a web application's caching mechanisms to inject malicious content into a site's cache, which can then be served to users who visit the site. The goal of a web cache poisoning attack is to trick users into executing arbitrary code or divulging sensitive information.

While potentially possible, this type of attack is hard to test for and it would require that the attacker has full access to the browser cache of the victim's machine. So this problem is more a problem for the 3rd party Dapp sites where they could be vulnerable to cache poisoning attacks if loaded via web caches, etc.

Clipboard tempering through malware

Clipboard tampering attacks are a type of cyber attack that exploit the clipboard feature in a user's operating system to steal sensitive information or inject malicious content into a user's device. When the user copies something to their clipboard, such as a password, cryptocurrency wallet address, or other sensitive information, the attacker's code intercepts and modifies the clipboard data and replaces the legitimate data with their own malicious content in this case the wallet address with their own wallet address, which can result in the user unknowingly sending cryptocurrency to the attacker.

These attacks are possible in the Dapp integration tests, but the user still has to verify the transaction inside the wallet before signing so this protection is deemed adequate to protect the end users against these sorts of attacks.

Malicious extension download - allowing code injection

This is still a likely scenario that someone clones the wallet and modifies the code of the original repo and implements additional code to capture user secrets. This is something that has been plaguing other wallet extensions and is a likely scenario for an attack against this extension as well. Here, we recommend that the developers continuously monitor the app stores in chrome and other targeted browsers to quickly and swiftly detect malicious versions of the app and thereby swiftly be able to identify and take down the malicious software before it can cause harm to the end users. However this is an indirect risk and not something that the extension can protect against directly.

Brute force attacks of password using OSINT to decrypt the encrypted private key.

An offline brute force attack is a type of cyber attack in which an attacker attempts to crack a password or encryption key by repeatedly guessing different combinations of characters until the correct one is found. Unlike online brute force attacks, which rely on attempting to guess passwords or keys using a live connection to a server, offline attacks typically involve downloading a file or database containing encrypted passwords, and then using specialized software to test every possible combination of characters until the correct password or key is found.

Our tests indicate that this is a likely attack against the end users of the system, but the extension has implemented an encryption algorithm that is well suited to make such attacks time consuming and expensive for the attacker.

General observations

Based on the results of the security audit, it can be concluded that the Lace Wallet browser extension is securely implemented and does not leak any user secrets. The wallet implements appropriate security measures to ensure the confidentiality and integrity of user secrets. The wallet can be considered secure for use by users who need to store and manage their crypto assets.

Leakage of key material and private keys **(No findings)**

FYEO has implemented automated tests to detect the leakage of any private or confidential key material or password. The tests were performed by the following steps:

1. Remote controlling the browser in a virtual machine via a chrome driver framework
2. Navigating to the browser extension and creating a new wallet from seed phrase
3. Dumping the memory of the virtual machine
4. Analyzing the memory dump with regular expressions and making sure that after locking the wallet no trace can be found of
    a. Private key in binary nor text form
    b. Encryption key material for locally stored private key
    c. Password used by the created wallet
    d. Mnemonic phrase used to initiate the wallet

Dapp integration tests

FYEO has written several scripts to test the functionality of the Dapp integration between the browser and the wallet extension. In general, the wallet extension is conforming to the standards as specified.

Accessing the extension from iframes and clickjacking

No findings

Several other browsers' wallets have had vulnerabilities and problems when loading the wallet extension from an iframe that is injected into the current site. This extension, however, did not export the API interface to the user if loaded from inside an iframe.

Prototype pollution

No findings

Prototype pollution is a vulnerability in JavaScript that can lead to unexpected or malicious behavior in web applications. It occurs when an attacker is able to modify the prototype object of a constructor function, thereby altering the behavior of all objects created from that constructor. This can lead to a wide range of attacks, including cross-site scripting (XSS), privilege escalation, and denial of service.

All of the test cases written for these attacks failed since the application is written in Typescript and all input data that passes the security barrier between the tab context and the extension context were securely implemented using Typescript

Looping Initiate trust of wallet through malicious website

Severity: Medium

FYEO found that the wallet extension repeatedly pops up for the same web page even if the user has explicitly canceled the trust initiation. This can lead to a vulnerability if a malicious website or phishing site constantly opens the extension trust modal until the user accepts the connection.

Fraudsters often use popups that loop until the user takes the desired action as a way to trick users into performing certain actions, such as downloading malware or providing sensitive information. These pop-ups can be extremely persistent and difficult to close, which can make users feel trapped and more likely to take the desired action.

If the user attempts to close the popup, the fraudster may use the identified vulnerability to make the popup reappear immediately after being closed.

Recommendations

We recommend that the application implements protection against these kinds of attacks. Several competing wallets have implemented protection against these types of phishing and malware attacks by implementing either blacklists and or keeping a state inside the app that keeps track of which sites the users have already declined an trust request.

Possible to initiate trust of wallet from non ssl encrypted sites

Severity: Low

FYEO has found that it is possible to initiate a trust of the wallet from a non SSL encrypted website.

Sending sensitive data to a non-SSL encrypted website can have serious security implications. When data is transmitted over the internet, it is susceptible to interception and eavesdropping. Without SSL encryption, the data can be read and even modified by anyone who can intercept the communication, such as attackers or malicious actors.

While being a useful feature for development and testing, this is a high risk if the trust is established on a production network where transactions have monetary value.

Recommendations

We recommend that the application checks whether the site is https encrypted and if not and the wallet is connected to the main net and not the pre production or the test network that the extension at least show an extra warning modal explaining to the user that this is very high risk associated with trusting a non encrypted site on the production network

Possible to initiate trust from private ip addresses and localhost

Severity: Low

Assigning trust to private IP address ranges can pose several security problems, Private IP address ranges are commonly used within local area networks and are not intended to be used as a means of authentication or authorization.

Lack of Authentication and Authorization: Using private IP address ranges as a means of authentication or authorization does not provide any form of real authentication or authorization since there are multiple addresses in existence.

Internal addresses are also Vulnerable to ip spoofing and arp cache poisoning on the local network especially if not encrypted with https.

This means that anyone with access to the network can potentially access the resources or services and exploit the wallet trust barrier to the Dapp.

Recommendations

We recommend that the application checks whether the site is a private ip and if and the wallet is connected to the main net and not the pre production or the test network that the extension at least show an extra warning modal explaining to the user that this is very high risk associated with trusting a non public site on the production network

Testing Methodology and framework

The testing methodology and framework are published under the the FYEO gitlab repository and will be cloned at the end of the assignment to into a new repository provided by the client

https://gitlab.com/f.y.e.o/automated-extension-testing

Overview of the framework and workflow

The framework functions by the following pipeline

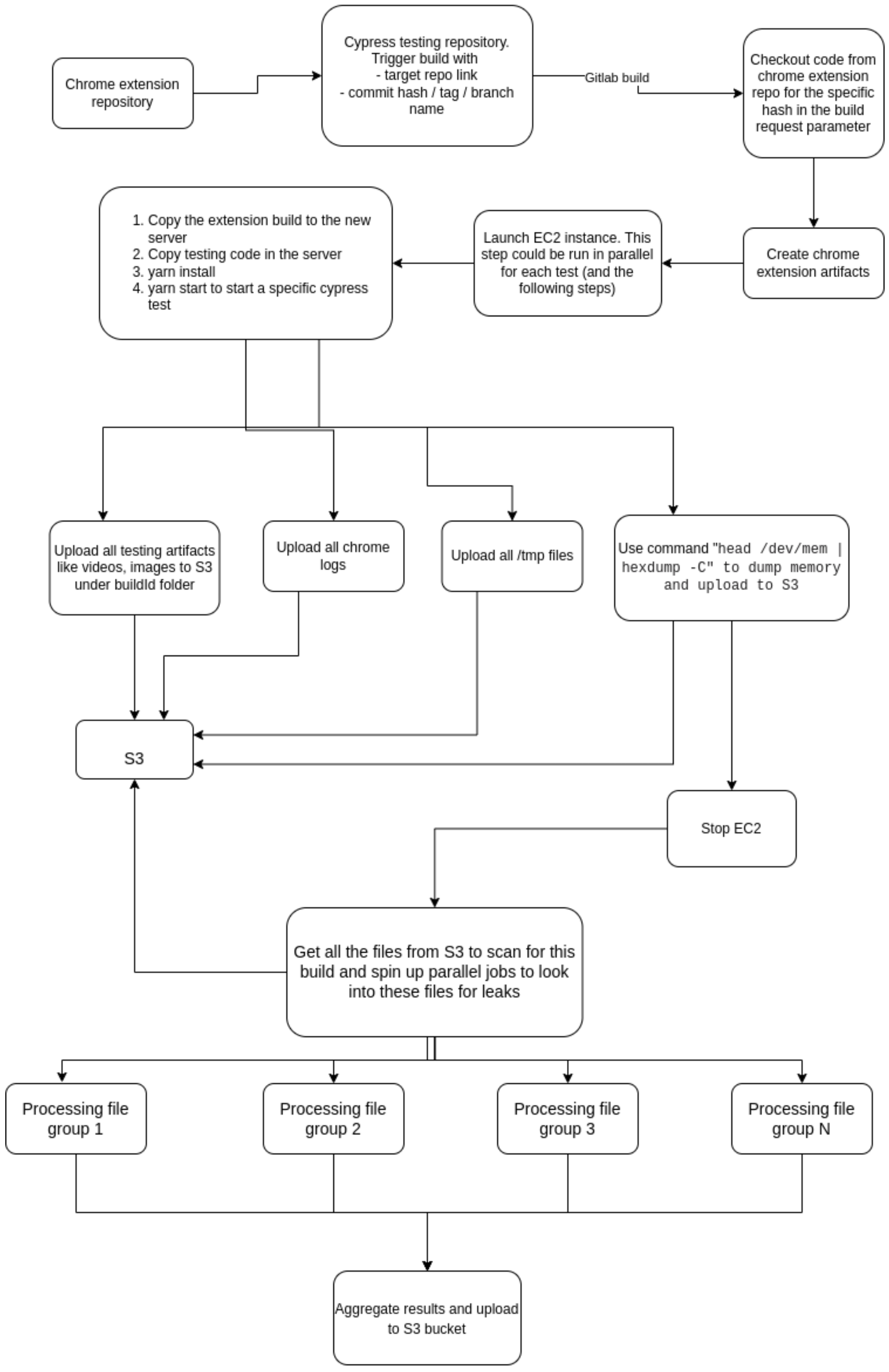
1. Set global parameters needed to build the extension
2. Build the extension
3. Starts a virtual machine
4. Starts a remote chrome browser with the extension loaded
5. Executes the test scripts through nightwatch
6. Collects evidence from the execution environment including log files and memory dumps
7. Generate issues in gitlab if any issue is encountered within the test

More details about the testing framework and how to operate it can be found in the readme.md file in the repo.

https://gitlab.com/f.y.e.o/automated-extension-testing#how-does-this-work

All the hardware wallet tests that were originally out of scope were not possible to perform using the automated system since the wallet needed physical interaction. I.e pin codes and due to the fact that USB devices could not be connected to the virtual machines remotely. Therefore these tests were performed locally using a raspberry pi and a SSH connection and manual triggering of the memory dump etc.

Gitlab Testing Pipeline


Chrome extension repository
Cypress testing repository. Trigger build with
- target repo link
- commit hash / tag / branch name
Gitlab build
Checkout code from chrome extension repo for the specific hash in the build request parameter
Create chrome extension artifacts
Launch EC2 instance. This step could be run in parallel for each test (and the following steps)
1. Copy the extension build to the new server
2. Copy testing code in the server
3. yarn install
4. yarn start to start a specific cypress test
Upload all testing artifacts like videos, images to S3 under buildId folder
Upload all chrome logs
Upload all /tmp files
Use command "head /dev/mem | hexdump -C" to dump memory and upload to S3
S3
Stop EC2
Get all the files from S3 to scan for this build and spin up parallel jobs to look into these files for leaks
Processing file group 1
Processing file group 2
Processing file group 3
Processing file group N
Aggregate results and upload to S3 bucket