Penetration tests of MorphoDAO FrontEnd for MorphoDAO

Report

document version: 1.0

document ID: SCRNG-1645

author: Dawid Pastuszak

test time period: 2022-03-28 – 2022-04-01

report date: 2022-04-04

TABLE OF CONTENTS

1. EXECUTIVE SUMMARY

1.1. Testing overview

The security tests of MorphoDAO FrontEnd were meant to verify whether the proper security mechanisms were in place to prevent unauthorized users from accessing the client's data and infrastructure and to detect the vulnerabilities which could cause financial losses to the client or their customers.

Security tests were performed using the following methods:

- Penetration testing – simulated attacks on MorphoDAO FrontEnd from the perspective of a standard user.

1.2. Summary of test results

- During the penetration testing, no vulnerabilities with critical risk impact were found.
- 1 vulnerability with low risk impact was identified:
  - Infura API key takeover (F1).
- Additionally, 4 recommendations have been proposed that do not have any direct risk impact. However, it is suggested to implement them due to good security practices.

2. SUMMARY OF IDENTIFIED VULNERABILITIES

2.1. Risk classification

Vulnerabilities	
Risk impact	**Description**
Critical	Vulnerabilities that affect the security of the entire system or all users. It is recommended to take immediate mitigating actions or limit the possibility of vulnerability exploitation.
High	Vulnerabilities that can lead to a significant financial loss including a confidential data leak, escalation of privileges, or a denial of service. It is recommended to take mitigating actions as soon as possible.
Medium	Vulnerabilities that affect the security of more than one user. They might require user interaction (e.g. as a result of an effective social-engineering attack), specific privileges, or custom prerequisites (e.g. access to client's internal network). The mitigating actions should be taken after eliminating the vulnerabilities with critical and high risk impact.
Low	Vulnerabilities that affect the security of individual users. They require custom prerequisites (e.g. user interaction or specific privileges). The mitigating actions should be taken after eliminating the vulnerabilities with critical, high, and medium risk impact.
Recommendations	
Methods of increasing the security of the system by implementing good security practices or eliminating weaknesses. No direct risk impact has been identified. The decision whether to take mitigating actions should be made by the client.	

2.2. Identified vulnerabilities

Vulnerability	Risk impact
SCRNG-1645-F1 Hardcoded API KEY in source code	Low
Recommendations	
SCRNG-1645-R1 Implement HTTP Secure Headers	
SCRNG-1645-R2 Missing Clickjacking attack protection	
SCRNG-1645-R3 Implement the Content Security Policy security header	
SCRNG-1645-R4 Do not load external resources	

3. PROJECT DESCRIPTION

3.1. Basic information

Testing team	Dawid Pastuszak Mateusz Olejarka Kamil Migdał
Testing time period	2022-03-28 – 2022-04-01
Report date	2022-04-01 – 2022-04-04
Document ID	SCRNG-1645
Document version	1.0

3.2. Target in scope

The object being analysed was MorphoDAO FrontEnd accessible from the URL address listed below:

- http://aave.morpho-labs.com.s3-website.eu-west-3.amazonaws.com/?network=mumbai
- https://develop-aave.herokuapp.com/
- https://aave.morpho-labs.com/?network=mumbai

The tests were performed in the test environment.

3.3. Threat analysis

The key threats were identified as follows:

- Modification of application source code
- Unauthorized access to other users' data
- Unauthorized modification of other users' data or traffic

3.4. Methodology

The testing team applied the methodology of white-box penetration tests. A penetration test is a controlled attempt to break through security controls applied in a particular application/system. In a white-box test, the testing team has access not only to the set of information as a typical user of the tested system, but also to the source code of the application. The tests were aimed at identification of vulnerabilities occurring in the application and defining possible attack scenarios conducted with techniques typical for attacks on web applications.

The report utilizes OWASP Application Security Verification Standard (ASVS) 4.0 and Common Vulnerability Scoring System (CVSS) 3.1.

3.5. Scope

Following the specification, the tests covered:

- A full range of security tests without any initial privileges,
- Tests performed as a member of standard user groups – to identify vulnerabilities resulting in key threats,
- Compliance with the OWASP Application Security Verification Standard (ASVS) 4.0, level 2.

4. LIST OF PERFORMED TESTS

4.1. Web application security testing

1. Security assessment of authentication process:
   - Reconnaissance and attempts to bypass or abuse the authentication mechanism,
2. Attempts to perform a Request Smuggling attack.
3. Verification of secure HTTP headers presence (Strict-Transport-Security, X-Content-Type-Options, Referrer-Policy, X-Frame-Options, Content-Security-Policy).
4. Verification of cache headers configuration.
5. Attempts to perform a Cross-Site Request Forgery attack.
6. Security analysis of SSL/TLS configuration.
7. Searching for sensitive or excessive information (in HTML comments, error messages, HTTP headers).
8. Performing a directory brute-force attack in order to find sensitive or excessive files and directories.
9. Verification if the libraries used by the application have any known vulnerabilities.
10. Checking for presence of typical web applications vulnerabilities (attempts to perform attacks like SQL Injection, Cross-Site Scripting, XML External Entity, Open Redirect Remote Code Execution, etc.).
11. Verification of the possibility to perform a DoS attack without any initial privileges.

5. VULNERABILITIES

F1. Hardcoded API KEY in source code

Risk impact	Low	CVSS	2.7	ASVS	V8
Exploitation conditions	Access to source code repository.				
Exploitation results	Infura API key takeover.				
References	How to: Use Data Protection https://docs.microsoft.com/en-us/dotnet/standard/security/how-to-use-data-protection Spring.IO: Managing secrets with Vault https://spring.io/blog/2016/06/24/managing-secrets-with-vault				
Remediation	When storing encrypted secrets inside configuration files, encryption keys should be stored as environmental variables.				

Vulnerability description:

API KEY were found in configuration files during code review. Key can be obtained by anyone with access to the source code and can be used to use Infura API.

Test case:

The following secret was found during code review:

- INFURA_KEY found in morpho-dashboard/config/index.ts file

Additionally test private key was sent in zip to SecuRing together with .env file.

- PRIVATE_KEY found in morpho-js/.env file.

6. RECOMMENDATIONS

R1. Implement HTTP Secure Headers

Description:

It is recommended to implement the following HTTP headers:

- HTTP Strict Transport Security (HSTS) – defines the timeframe in which the browser can connect to the webserver only through the HTTPS protocol. The lack of the HSTS header may expose the application to protocol downgrade attack and cookie hijacking.
- Referrer-Policy – controls what information should be sent in Referrer header in HTTP requests.
- X-Content-Type-Options: nosniff – prevents browsers from trying to determine the correct MIME type of the response.

How to implement:

Implement the security headers.

HTTP Strict Transport Security (HSTS)

A Strict-Transport-Security HTTP header should be sent with each HTTPS response. The syntax is as follows:

```
Strict-Transport-Security: max-age=<seconds>[; includeSubDomains]
```

The parameter *max-age* gives the time frame in seconds for requirement of HTTPS and should be chosen quite high, e.g. several months. The flag *includeSubDomains* defines that the policy also applies to subdomains.

Referrer-Policy

The *Referrer-Policy* header with the appropriate directive should be added to each HTTP response, e.g.:

- *no-referrer* – the browser will not send any referrer information,
- *same-origin* – the browser will send referrer information only for same-site origins.

X-Content-Type-Options: nosniff

The following HTTP header should be set at least in all responses which contain user input:

```
X-Content-Type-Options: nosniff
```

References:

OWASP Secure Headers Project
https://owasp.org/www-project-secure-headers/

WEB.DEV Security Headers Quick Reference
https://web.dev/security-headers/

R2. Missing Clickjacking attack protection

Description:

Application is not protected against Clickjacking attacks, which occur when an attacker places the vulnerable site inside an iframe. If a victim logged into the application opens an attacker-controlled site, it is possible to perform actions on the target website.

Every action must be additionally confirmed inside MetaMask browser's extension, therefore there is no practical usage of vulnerability, but it is a good practice to implement Clickjacking protections.

How to implement:

Prevent the browser from loading the page in a frame by using Content Security Policy with frame-ancestors directive or X-Frame-Options HTTP headers.

References:

CAPEC-103: Clickjacking
https://capec.mitre.org/data/definitions/103.html

OWASP Clickjacking
https://owasp.org/www-community/attacks/Clickjacking

OWASP Clickjacking Defense Cheat Sheet
https://cheatsheetseries.owasp.org/cheatsheets/Clickjacking_Defense_Cheat_Sheet.html

Content Security Policy
https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Content-Security-Policy

OWASP Content Security Policy Cheat
Sheethttps://cheatsheetseries.owasp.org/cheatsheets/Content_Security_Policy_Cheat_Sheet.html

R3. Implement the Content Security Policy security header

Description:

Content Security Policy (CSP) is a browser security mechanism that helps to detect and mitigate certain types of attacks, including Cross Site Scripting (XSS). The mechanism instructs the browser - using a standardized set of directives - what resources are allowed to load for a page. The application does not use the mechanism.

How to implement:

It is recommended to implement Content Security Policy which introduces another layer of protection against the classical reflected and stored XSS.

References:

Securing – Why should you care about Content Security Policy?
https://www.securing.pl/en/why-should-you-care-about-content-security-policy/

Content Security Policy
https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Content-Security-Policy

OWASP Content Security Policy Cheat Sheet
https://cheatsheetseries.owasp.org/cheatsheets/Content_Security_Policy_Cheat_Sheet.html

Adopting CSP
https://csp.withgoogle.com/docs/adopting-csp.html

CSP Evaluator
https://csp-evaluator.withgoogle.com/

WEB.DEV Security Headers Quick Reference
https://web.dev/security-headers/

R4. Do not load external resources

Description

Loading external resources such as img or svg, may be concerning from user privacy perspective and open possibilities for other vulnerabilities like XSS.

How to implement:

Store all necessary files on MorphoDAO servers.

References:

Allowing images from external sources opens doors to serious security exploits and privacy risks:
https://meta.stackexchange.com/questions/55665/allowing-images-from-external-sources-opens-doors-to-serious-security-exploits-a/279664

7. TERMINOLOGY

This section explains the terms that are related to the methodology used in this report.

Risk = Threat + Vulnerability

Threat

Any circumstance or event with the potential to adversely impact organizational operations (including mission, functions, image, or reputation), organizational assets, or individuals through an information system via unauthorized access, destruction, disclosure, modification of information, and/or denial of service.[1]

Vulnerability

Weakness in an information system, system security procedures, internal controls, or implementation that could be exploited or triggered by a threat source.[1]

Risk

The level of impact on organizational operations (including mission, functions, image, or reputation), organizational assets, or individuals resulting from the operation of an information system given the potential impact of a threat and the likelihood of that threat occurring.[1]

The risk impact can be estimated based on the complexity of exploitation conditions (representing the likelihood) and the severity of exploitation results.

		Complexity of exploitation conditions		
		Simple	Moderate	Complex
Severity of exploitation results	Major	Critical	High	Medium
	Moderate	High	Medium	Low
	Minor	Medium	Low	Low

[1] NIST FIPS PUB 200: Minimum Security Requirements for Federal Information and Information Systems. Gaithersburg, MD: Computer Security Division, Information Technology Laboratory, National Institute of Standards and Technology.

8. CONTACT

Person responsible for providing explanations:

Dawid Pastuszak
e-mail: dawid.pastuszak@securing.pl
tel.: +48 12 425 25 75
mob.: +48 502 745 328

http://www.securing.pl
e-mail: info@securing.pl
Kalwaryjska 65/6
30-504 Kraków
tel./fax.: +48 (12) 425 25