Contract Audit Results

Prepared on: 03/09/2024

Contract: AUD532

Prepared by:

Charles Holtzkampf

Sentnl

Prepared for:

CoverCrypto

Table of Contents

Executive Summary

During this security audit, in addition to following OWASP guidelines to test for common vulnerabilities, we were specifically asked by CoverCrypto to verify whether CoverCrypto has the ability to spend or move crypto assets belonging to their users on the platform

We can confirm that it would be impossible for CoverCrypto to move your cryptoassets. The application utilises Vezgo, which is a read-only connector for your crypto accounts. Vezgo ensures that while we can access and view your crypto data (for the purposes of proving ownership), we cannot make any transactions on your behalf, or suggest any transactions for you to approve.

Below outlines any issues found during the audit of the web application.

- 9bbb5d6196b696375322181dd1c98f7fd04cea60

- The contract has 0 remarks.
- 5 Low security issues were found and resolved.
- 0 Medium security issues were found.
- 0 High security issues were found.
- The risk associated with this application is low.

REMARK	LOW	MEDMIM	HIGH
0	5	0	0

Severity Description

REMARK

Remarks are instances in the code that are worthy of attention, but in no way represent a security flaw in the code. These issues might cause problems with the user experience, confusion with new developers working on the project, or other inconveniences.

Things that would fall under remarks would include:

- Instances where best practices are not followed
- Spelling and grammar mistakes
- Inconsistencies in the code styling and structure

LOW

Issues of Low severity can cause problems in the code, but would not cause the code to crash unexpectedly or for funds to be lost. It might cause results that would be unexpected by users, or minor disruptions in operations. Low problems are prone to become medium problems if not addressed appropriately.

Things that would fall under low would include:

- Logic flaws.
- Code duplication.
- Ambiguous code.

MEDIUM

Issues of medium security can cause the code to crash unexpectedly or denial service attacks.

Things that would fall under medium would include:

- Logic flaws that cause crashes.
- Denial of service attacks.
- Broken access control.

HIGH

High issues can cause a complete system compromise or account compromise.

Things that would fall under critical would include:

- Missing checks for authorisation.
- Account takeovers.
- SQL injection.
- Any exploit that could severely comprise security.

Methodology

Throughout the review process, we check that the token contract:

- Documentation and code comments match logic and behaviour
- Is not affected by any known vulnerabilities

Our team follows best practices and industry-standard techniques to verify the proper implementation of the web application. Our secuity auditors reviewed the code line by line, documenting any issues as they were discovered. Ontop of the line by line review, we also perform web penetration testing against the web application..

Our strategies consist largely of manual collaboration between multiple team members at each stage of the review, including:

I. Due diligence in assessing the overall code quality of the codebase.
II. Testing contract logic against common and uncommon attack vectors.
III. Thorough, manual review of the codebase, line-by-line.

Our testing includes:

- **Injection Flaws**
- **Broken Authentication**
- **Sensitive Data Exposure**
- **XML External Entities (XXE)**
- **Broken Access Control**
- **Security Misconfiguration**
- **Cross-Site Scripting (XSS)**
- **Insecure Deserialization**

LOW

1 Password reset token expiry set to default 24 hours

The function GeneratePasswordResetTokenAsync is what controls the password resets and relies on the DataProtectionTokenProviderOptions class.

```
Application/Users/Commands/SendPasswordResetEmailCommand.cs
await _userManager.GeneratePasswordResetTokenAsync(user);
```

In ASP.NET the default expiry length for password reset tokens is set in the DataProtectionTokenProviderOptions class, which by default is set to 24 hours.

https://learn.microsoft.com/en-us/dotnet/api/microsoft.aspnetcore.identity.dataprotectiontokenprovider 8.0

Recommendations:

Override the default 24 hours in Program.cs to like 1 hour.

```
builder.Services.Configure<DataProtectionTokenProviderOptions>(options =>
{
    options.TokenLifespan = TimeSpan.FromHours(1); // Example: Set token lifespan to 1
↪  hours
});
```

2 No Rate Limit on password reset requests(Leads to huge email flooding/email bombing)

No rate limiting has been set for generating password reset emails for accounts. Having the ability to perform countless password resets, could allow an attacker to collect 1000s of password reset tokens and run this through a entropy tool to analyze the quality of randomness. Although unlikely that there is a problem with the randomness of the tokens, the possibility is there. Our suggestion would be to enable rate limiting to prevent countless password reset requests.

Recommendations:

Configure Rate limiting on the web server level (assuming AWS provides this) or implement IP rate limiting directly in .NET.

3 X-Frame-Options enabled, but better to have a CSP Policy

We found a potential clickjacking vulnerability on the site. However, the headers you provided include the X-Frame-Options: SAMEORIGIN directive, which helps prevent clickjacking by allowing the content to be framed only by pages from the same origin. This is a good measure to mitigate clickjacking attacks.

However, the headers presented by the site are missing a Content Security Policy (CSP), which would be recommended to further protect your site from clickjacking and other potential threats.

Recommendations:

While X-Frame-Options: SAMEORIGIN is a strong measure against clickjacking, it is also advisable to implement a Content Security Policy (CSP) for a more comprehensive defense. Here's an example of what a CSP header might look like to further enhance security:

```
Content-Security-Policy: default-src 'self'; frame-ancestors 'self';
```

4 SaveConnection method allows client side interception and manipulation

The SaveConnection method currently processes data provided by the client, including account details and provider information, and saves this data to the database. Since this data is transmitted from the client side, certain fields like are vulnerable to interception and manipulation. An attacker could modify values such as Provider, Connection ID before the data is sent to the server, potentially causing unauthorized data to be saved or manipulated in your system.

```
Web/Areas/Connections/Controllers/Api/ConnectionsApiController.cs

[HttpPost]
[Route("SaveConnection")]
public async Task<IActionResult> SaveConnection(AccountDto account)
......
```

Recommendations:

Implement server side lookups for these values and prevent client side manipulation.

5 Wallet addresses are saved in DB as text

While wallet addresses are verified against the listings on Vezgo, and therefore do not present an immediate security risk, we recommend applying a encryption function to these database fields. This will further safeguard them against potential manipulation at the database level.

Recommendations:

Implement AES using a secure key and initialization vector to encrypt wallet addresses before storing them in the database.