BlockApex Penetration Testing Service Report for Lightlink

Version

1.0.0

Table of Content

1. Executive Summary

1.1 Report Objectives

This document details the security assessment (vulnerability assessment and penetration testing) of Lightlink Dapp The purpose of the assessment was to provide a review of the security posture of example.com App infrastructure, as well as to identify potential weaknesses in its infrastructure.

1.2 Scope of Work

Type	Details
Target System Name	Lightlink Dapp
Target System URL(s)	https://beta.lightlink.io/bridge?typeBridge=deposit&tokenBridge=ETH
Intrusive Tests	Yes
Type of Test	Black Box Test

1.4 Summary of Findings

Severity	*Name*	*Impact*	*Status*
Low	Missing HTTP Security Headers (Section 2.1)	It makes the website less secure, potentially allowing harmful actions like tricking users into clicking on things they shouldn't.	Fixed
Low	Exposure of Sensitive Keys in JS Files (Section 2.2)	It risks unauthorized access or misuse of services connected to the exposed keys, potentially compromising user data.	Fixed
None	Automated Testing	Overall system	Acknowledged

1.5 Summary of Recommendations

- Implement critical security headers, namely Content-Security-Policy (CSP), X-Frame-Options, X-XSS-Protection, and Referrer-Policy, into your HTTP responses. Also, correct the configuration of the existing Strict-Transport-Security (HSTS) to enforce secure communication.

- Sensitive keys should not be stored within client-side code. Move these keys server-side and securely handle client-server communications involving these keys using environment variables or similar secure methods. Additional security measures such as IP whitelisting and regular key rotation are also recommended.

2. Detail Findings - Technical Details

2.1 Vulnerability # 1

Missing HTTP Security Headers

2.1.1 Description

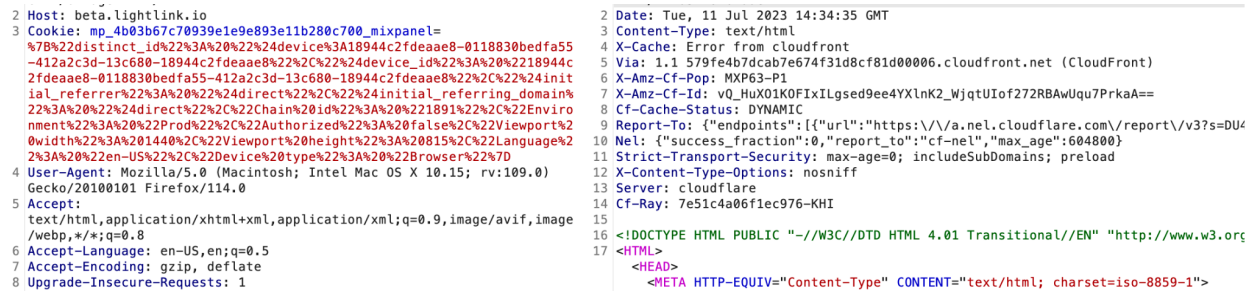
The HTTP response headers for the application are missing critical security headers, namely Content-Security-Policy (CSP), X-Frame-Options, X-XSS-Protection, and Referrer-Policy. Also, the existing Strict-Transport-Security (HSTS) is incorrectly configured.

Likelihood	Severity	Affected Endpoint
High	low	https://beta.lightlink.io/bridge

2.1.2 Steps To Reproduce

1. Make a GET request to the endpoint (https://beta.lightlink.io/bridge).
2. Observe the response headers.
3. Note the absence of the aforementioned security headers

Screen Shot



```
Host: beta.lightlink.io
Cookie: mp_4b03b67c70939e1e9e893e11b280c700_mixpanel=
%7B%22distinct_id%22%3A%20%22%24device%3A18944c2fdeaae8-0118830bedfa55
-412a2c3d-13c680-18944c2fdeaae8%22%2C%22%24device_id%22%3A%20%2218944c
2fdeaae8-0118830bedfa55-412a2c3d-13c680-18944c2fdeaae8%22%2C%22%24init
ial_referrer%22%3A%20%22%24direct%22%2C%22%24initial_referring_domain%
22%3A%20%22%24direct%22%2C%22Chain%20id%22%3A%20%221891%22%2C%22Enviro
nment%22%3A%20%22Prod%22%2C%22Authorized%22%3A%20false%2C%22Viewport%2
0width%22%3A%201440%2C%22Viewport%20height%22%3A%20815%2C%22Language%2
2%3A%20%22en-US%22%2C%22Device%20type%22%3A%20%22Browser%22%7D
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.15; rv:109.0)
Gecko/20100101 Firefox/114.0
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image
/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Upgrade-Insecure-Requests: 1
```

```
Date: Tue, 11 Jul 2023 14:34:35 GMT
Content-Type: text/html
X-Cache: Error from cloudfront
Via: 1.1 579fe4b7dcab7e674f31d8cf81d00006.cloudfront.net (CloudFront)
X-Amz-Cf-Pop: MXP63-P1
X-Amz-Cf-Id: vQ_HuXO1KOFIxILgsed9ee4YXlnK2_WjqtUIof272RBAwUqu7PrkaA==
Cf-Cache-Status: DYNAMIC
Report-To: {"endpoints":[{"url":"https:\/\/a.nel.cloudflare.com\/report\/v3?s=DU
Nel: {"success_fraction":0,"report_to":"cf-nel","max_age":604800}
Strict-Transport-Security: max-age=0; includeSubDomains; preload
X-Content-Type-Options: nosniff
Server: cloudflare
Cf-Ray: 7e51c4a06f1ec976-KHI

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org
<HTML>
  <HEAD>
    <META HTTP-EQUIV="Content-Type" CONTENT="text/html; charset=iso-8859-1">
```

2.1.3 Impact

The absence of these headers leaves the application vulnerable to:

1. XSS attacks: If CSP is not defined, the application can become an easy target for XSS attacks.
2. Clickjacking: Without the X-Frame-Options header, attackers can load your website inside an iframe and deceive users into clicking elements.
3. Insecure communication: With HSTS set to a max-age of 0, the application can be exposed to SSL stripping attacks, making it possible for attackers to downgrade secure HTTPS requests to HTTP.

2.1.4 Mitigation

1. Implement the CSP header to specify trusted sources of scripts, styles, and other resources.
2. Implement the X-Frame-Options header with a value of "SAMEORIGIN" or "DENY" to prevent clickjacking.
3. Implement the X-XSS-Protection header with a value of "1; mode=block" to activate the XSS protection mechanism in the user's web browser.
4. Implement the Referrer-Policy header with a value suitable to your application's privacy needs.
5. Set the Strict-Transport-Security header with a suitable max-age (preferably at least a year) to enforce all communication via HTTPS.

2.1.5 Reference

- OWASP
- Metamask Docs
- Embark Docs

2.2 Vulnerability # 2

Exposure of Sensitive Keys in JavaScript Files

2.2.1 Description

Several Square Access Tokens and Heroku API keys were found openly exposed within a JavaScript file (main.a0c96d31.js) in your application. Although the exposed keys were reported as non-vulnerable, storing sensitive keys within client-side code is a bad practice and should be avoided. This information could be potentially used by malicious actors to attempt unauthorized access or misuse of the associated services.

Likelihood	Severity	Affected Endpoint
high	Low	https://beta.lightlink.io/static/js/main.a0c96d31.js

2.2.2 Steps To Reproduce

1. Navigate to https://beta.lightlink.io/static/js/main.a0c96d31.js.
2. View the source of the JavaScript file.
3. The exposed keys can be found embedded within the JavaScript code.

2.2.3 Impact

If the keys are active, an attacker could:

- Access or modify data associated with your application on Heroku.
- Interact with the Square API, possibly initiating unauthorized transactions.

2.2.4 Mitigation

- Keys and other sensitive information should never be stored in client-side code. Instead, they should be stored server-side, and any client-server communication involving these keys should be securely handled on the server-side.
- Environment variables could be used to store these sensitive keys on the server side.
- For Heroku, you can use Config Vars to store sensitive information. You can set them in the settings of your Heroku app.
- For Square, consider using a backend server to securely store and manage your access tokens.
- Implement additional security measures such as IP whitelisting, where only specific, trusted IP addresses can access your API endpoints.
- Regularly rotate and change keys and tokens.

2.2.5 Reference

- OWASP - Sensitive Data Exposure
- Heroku Configuration and Config Vars
- Square Access Tokens

Automated Testing

Automated testing is a crucial aspect of ensuring the security of any web application. This process involves the use of automated tools to identify security vulnerabilities and weaknesses. It provides a systematic approach to test for common vulnerabilities and exposes areas where the application does not comply with established security standards.

One of the popular automated testing tools is Nessus, which we used in our assessment of the Light Link Bridge DApp. Nessus is a comprehensive vulnerability scanner, known for its accuracy and extensive database of vulnerabilities.

Nessus helps identify:

- Misconfigurations and missing patches
- Unsafe practices in software development
- Compliance with security standards

Through automated testing with Nessus, we can provide a comprehensive view of vulnerabilities, reduce manual effort, and speed up the remediation process.

Please find the Nessus-generated report for the Light Link Bridge DApp at the following Link

Disclaimer

The application provided for security assessment has been reviewed using methodologies to date, and there are cybersecurity vulnerabilities and flaws in the application source code, which are documented in this report (Source Code); the Source Code compilation, deployment, and functionality (performing the intended functions).

The assessment makes no claims or guarantees about the code's security. Furthermore, it cannot be deemed a sufficient evaluation of the code's efficiency and safety, bug-free status, or any other safety claims. While we did our best to conduct the analysis and produce this report, it is essential to mention that you should not solely rely on it — To ensure security, we recommend conducting many independent audits and launching a public bug bounty programme.

Any web or mobile application is developed, deployed and maintained on a particular platform. The platform, server, its programming language and any other software or application related to it can have vulnerabilities that can lead to attacks or hacks. Thus, our audit can not guarantee the explicit security of the audited product.