QuillAudits

AUDIT REPORT

October 2025

For

omnyx

Table of Content

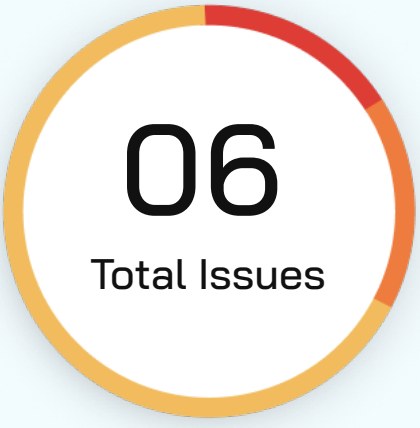
Executive Summary

Project Name	Omnyx
Protocol Type	Wallet Pentest
Source Code URL	https://wallet.omnyx.global/
Overview	Omnyx Wallet is a non-custodial, multi-chain cryptocurrency wallet designed for both beginners and advanced users. It offers a user-friendly interface that allows users to send, receive, buy, sell, and swap digital assets seamlessly. The wallet emphasizes security by encrypting user data and providing unique passkeys to unlock hidden subaccounts, enhancing privacy and control. Omnyx Wallet supports multiple blockchains, enabling users to manage various cryptocurrencies within a single platform.
Scope of the Audit	The scope of this pentest was to analyze the web app for quality, security ,and correctness
Review 1	25th Sept 2025 - 2nd Oct 2025
Updated Code Received	5th October 2025 and 22nd October 2025
Review 2	22nd October 2025

Verify the Authenticity of Report on QuillAudits Leaderboard:

https://www.quillaudits.com/leaderboard

Number of Issues per Severity



Critical	0(0.0%)
High	1 (16.6%)
Medium	1 (16.6%)
Low	4 (66.6%)
Informational	0(0.0%)

Severity

Issues	Critical	High	Medium	Low	Informational
Open	0	0	0	0	0
Acknowledged	0	0	0	**4**	0
Partially Resolved	0	0	0	0	0
Resolved	0	**1**	**1**	0	0

Summary of Issues

Issue No.	Issue Title	Severity	Status
1	Weak Encryption Key Derivation for LocalStorage Encryption	High	Resolved
2	Content Security Policy Allows ‘unsafe-eval’ and ‘unsafe-inline’	Medium	Resolved
3	Clickjacking	Low	Acknowledged
4	Overly Permissive CORS Policy Allowing Any Origin	Low	Acknowledged
5	Long-Lived Authentication Tokens Without Expiration	Low	Acknowledged
6	Known Vulnerable Dependencies with High Severity CVEs	Low	Acknowledged

Checked Vulnerabilities

- Improper Authentication
- Improper Resource Usage
- Improper Authorization
- Insecure File Uploads
- Insecure Direct Object References
- Client-Side Validation Issues
- Rate Limit
- Input Validation
- Injection Attacks
- Cross-Site Scripting (XSS)
- Cross-Site Request Forgery
- Security Misconfiguration
- Broken Access Controls
- Insecure Cryptographic Storage
- Insufficient Cryptography
- Insufficient Session Expiration
- Insufficient Transport Layer Protection
- Unvalidated Redirects and Forwards
- Information Leakage
- Broken Authentication and Session Management
- Denial of Service (DoS) Attacks
- Malware
- Third-Party Components

And More..

Techniques and Methods

Throughout the pentest of application, care was taken to ensure:

- Information gathering – Using OSINT tools information concerning the web architecture, information leakage, web service integration, and gathering other associated information related to web server & web services.
- Using Automated tools approach for Pentest like Nessus, Acunetix etc.
- Platform testing and configuration
- Error handling and data validation testing
- Encryption-related protection testing
- Client-side and business logic testing

Tools and Platforms used for Pentest:

Burp Suite	Acunetix	Nmap
DNSenum	Neucli	Metasploit
Dirbuster	Nabbu	Horusec
SQLMap	Turbo Intruder	Postman
Netcat	Nessus	And Many more..

Types of Severity

Every issue in this report has been assigned to a severity level. There are five levels of severity, and each of them has been explained below.

Critical: Immediate and Catastrophic Impact

Critical issues are the ones that an attacker could exploit with relative ease, potentially leading to an immediate and complete loss of user funds, a total takeover of the protocol's functionality, or other catastrophic failures. Critical vulnerabilities are non-negotiable; they absolutely must be fixed.

High (H): Significant Risk of Major Loss or Compromise

High-severity issues represent serious weaknesses that could result in significant financial losses for users, major malfunctions within the protocol, or substantial compromise of its intended operations. While exploiting these vulnerabilities might require specific conditions to be met or a moderate level of technical skill, the potential damage is considerable. These findings are critical and should be addressed and resolved thoroughly before the contract is put into the Mainnet.

Medium (M): Potential for Moderate Harm Under Specific Circumstances

Medium-severity bugs are loopholes in the protocol that could lead to moderate financial losses or partial disruptions of the protocol's intended behavior. However, exploiting these vulnerabilities typically requires more specific and less common conditions to occur, and the overall impact is generally lower compared to high or critical issues. While not as immediately threatening, it's still highly recommended to address these findings to enhance the contract's robustness and prevent potential problems down the line.

Low (L): Minor Imperfections with Limited Repercussions

Low-severity issues are essentially minor imperfections in the smart contract that have a limited impact on user funds or the core functionality of the protocol. Exploiting these would usually require very specific and unlikely scenarios and would yield minimal gain for an attacker. While these findings don't pose an immediate threat, addressing them when feasible can contribute to a more polished and well-maintained codebase.

Informational (I): Opportunities for Improvement, Not Immediate Risks

Informational findings aren't security vulnerabilities in the traditional sense. Instead, they highlight areas related to the clarity and efficiency of the code, gas optimization, the quality of documentation, or adherence to best development practices. These findings don't represent any immediate risk to the security or functionality of the contract but offer valuable insights for improving its overall quality and maintainability. Addressing these is optional but often beneficial for long-term health and clarity.

Types of Issues

Open

Security vulnerabilities identified that must be resolved and are currently unresolved.

Resolved

These are the issues identified in the initial audit and have been successfully fixed.

Acknowledged

Vulnerabilities which have been acknowledged but are yet to be resolved.

Partially Resolved

Considerable efforts have been invested to reduce the risk/impact of the security issue, but are not completely resolved.

Severity Matrix

Impact

Likelihood	High	Medium	Low
High	Critical	High	Medium
Medium	High	Medium	Low
Low	Medium	Low	Low

Impact

- **High** - leads to a significant material loss of assets in the protocol or significantly harms a group of users.
- **Medium** - only a small amount of funds can be lost (such as leakage of value) or a core functionality of the protocol is affected.
- **Low** - can lead to any kind of unexpected behavior with some of the protocol's functionalities that's not so critical.

Likelihood

- High - attack path is possible with reasonable assumptions that mimic on-chain conditions, and the cost of the attack is relatively low compared to the amount of funds that can be stolen or lost.
- Medium - only a conditionally incentivized attack vector, but still relatively likely.
- Low - has too many or too unlikely assumptions or requires a significant stake by the attacker with little or no incentive.

High Severity Issues

Weak Encryption Key Derivation for LocalStorage Encryption

Resolved

Description

The application utilizes CryptoJS AES.encrypt() with a passphrase-based key derivation function (PBKDF2 with only 1 iteration by default) for encrypting sensitive data stored in localStorage. Specifically, the file /src/lib/encryption.js uses 'AES.encrypt(data, key)' where the key is a hex-encoded token. CryptoJS, when provided a string password, defaults to OpenSSL-compatible PBKDF2 with only 1 iteration, which is cryptographically weak and highly vulnerable to brute-force attacks. Seeds, private keys, and cache data are encrypted using this vulnerable mechanism.

Vulnerable CODE

```
/src/lib/encryption.js line 11: return AES.encrypt(data, key).toString();
```

```
/src/lib/storage/ClientStorage.js line 56: item = encryption.encrypt(item,
hex.encode(token));
```

Recommendation

Replace CryptoJS with the WebCrypto API (SubtleCrypto) using AES-GCM for authenticated encryption. Implement proper key derivation using PBKDF2 with a high iteration count (e.g., 100,000+ iterations or Argon2). Alternatively, if the token is intended to be a 256-bit key, use it directly (as raw bytes) with crypto.subtle.importKey() and crypto.subtle.encrypt() in AES-GCM mode, rather than treating it as a password.

Medium Severity Issues

Content Security Policy Allows ‘unsafe-eval’ and ‘unsafe-inline’

Resolved

Description

The application’s Content Security Policy (CSP), located in /index.html (lines 9-40), explicitly permits ‘unsafe-eval’ and ‘unsafe-inline’ for the script-src directive. This configuration severely undermines the protective capabilities of CSP against Cross-Site Scripting (XSS) attacks. By allowing these directives, the policy permits the execution of inline scripts and eval()-based code, which are primary vectors for XSS exploitation.

Vulnerable CODE

Line 11: script-src ‘self’ ‘unsafe-eval’ ‘unsafe-inline’ data:;
Line 12: style-src ‘self’ ‘unsafe-inline’;

Proof: This CSP configuration allows:

Execution of eval() and Function() calls.
Execution of code within inline <script> tags.
Execution of inline event handlers (e.g., onclick=“alert(1)”).
Execution of scripts embedded via data: URIs.

Any XSS vulnerability, even a minor one, within the application can be fully exploited due to these overly permissive CSP directives, allowing an attacker to inject and execute arbitrary JavaScript code.

Recommendation

1. Remove ‘unsafe-eval’ from script-src: Find alternative, secure approaches for dynamic code execution (e.g., JSON parsing, WebAssembly, trusted types).
2. Remove ‘unsafe-inline’ from script-src and style-src: Implement a nonce-based or hash-based CSP for all inline scripts and styles.
3. Remove ‘data:’ from script-src: If data: URIs are necessary, restrict them to specific trusted MIME types.
4. Consider implementing strict-dynamic with nonces for modern browsers to allow dynamically added scripts to inherit trust.

Low Severity Issues

Clickjacking

Acknowledged

Description

The site at **https://target.com/** can be embedded inside an attacker-controlled iframe because the application does not consistently prevent framing. This enables a malicious page to visually hide or disguise the framed content and trick a user into clicking buttons or performing actions inside the framed application (UI redressing / clickjacking).

Impact

- Phishing / UX deception: An attacker can present the site inside a crafted page and trick authenticated users into taking actions they did not intend (e.g., clicking “Claim”, “Transfer”, approving wallet interactions).
- Unauthorized actions: If critical actions (payments, transfers, wallet binds) are available via a single click, users might unknowingly approve them while interacting with the attacker page.

Recommendation

Set a frame-ancestors CSP (preferred modern control)
Add the following HTTP response header (adjust allowed origins as needed):

POC

https://clickjacker.io/test?url=https://target.com/

Overly Permissive CORS Policy Allowing Any Origin

Acknowledged

Description

All API responses include the header Access-Control-Allow-Origin: *, which allows any website from any origin to make authenticated requests to the wallet API. This configuration completely bypasses the browser's same-origin policy security protection

Impact

Malicious websites can interact with user wallets without permission
Data exfiltration to unauthorized domains
Wallet transactions can be initiated from attacker-controlled sites
Phishing attacks become significantly easier to execute

Recommendation

IMMEDIATE: Restrict CORS to only trusted origins:

Access-Control-Allow-Origin: **https://target.com**

If multiple origins are needed, implement a whitelist approach:

```
const allowedOrigins = ['https://wallet.target.com', 'https://app.target.com'];
// Validate origin against whitelist before setting header
```

Remove Access-Control-Allow-Credentials: true when using wildcard origin (current misconfiguration)

Long-Lived Authentication Tokens Without Expiration

Acknowledged

Description

The registration endpoint returns 128-character hexadecimal tokens (deviceToken and walletToken) with no visible expiration mechanism, refresh capability, or token rotation. These tokens appear to provide persistent authentication and could remain valid indefinitely if compromised.

Response:


```
{
  "deviceToken":
"e5529d0501485e88f67f51413c73b8a3059ada2e38ba7833d7f3fa034d66548a0da1d213604159
f3745192f867d81fc2eb10f1e29983ce9767658f99577baca7",
  "walletToken":
"7a50a8fdbdfa48a0924fe0ac1a2f77a006e14d9274831b19e01027560b031630af2ea1b963091d
715070611d7c6e50c88d46e34cb30225456254c864a5b22aaf"
}
```


Impact

Stolen tokens can be used indefinitely for unauthorized access
No mechanism to invalidate compromised tokens
Increased window of opportunity for attackers

Recommendation

Implement token rotation on refresh
Store token metadata (device info, IP, last used) for security monitoring
Implement automatic token expiration on suspicious activity

Known Vulnerable Dependencies with High Severity CVEs

Acknowledged

Description

Multiple critical dependencies have known security vulnerabilities:
1. vue-i18n@9.2.2 - CVE-2024-52809: DOM-based XSS via escapeParameterHtml (GHSA-x8qp-wqqm-57ph)
2. axios@1.4.0 - CSRF vulnerability (GHSA-wf5p-g6vw-rhxx)
3. @sentry/browser@7.56.0 - Prototype pollution (GHSA-593m-55hh-j8gv)
4. crypto-js@4.1.1 - Multiple cryptographic weaknesses (not officially CVE'd but widely known)

Vulnerable Code

wallet_front/package.json

Recommendation

Update all vulnerable dependencies immediately

```
"vue-i18n": "^9.14.5",          // Fixes CVE-2024-52809 XSS vulnerability
"axios": "^1.7.7",              // Fixes GHSA-wf5p-g6vw-rhxx CSRF vulnerability
"@sentry/browser": "^7.119.1"   // Fixes GHSA-593m-55hh-j8gv prototype pollution
```

Closing Summary

In this report, we have considered the security of the Omnyx. We performed our audit according to the procedure described above.

Some issues of High, medium and low severity were found. The Omnyx team resolved a few of the issues mentioned, and others were acknowledged

Disclaimer

At QuillAudits, we have spent years helping projects strengthen their smart contract security. However, security is not a one-time event threats evolve, and so do attack vectors. Our audit provides a security assessment based on the best industry practices at the time of review, identifying known vulnerabilities in the received smart contract source code.

This report does not serve as a security guarantee, investment advice, or an endorsement of any platform. It reflects our findings based on the provided code at the time of analysis and may no longer be relevant after any modifications. The presence of an audit does not imply that the contract is free of vulnerabilities or fully secure.

While we have conducted a thorough review, security is an ongoing process. We strongly recommend multiple independent audits, continuous monitoring, and a public bug bounty program to enhance resilience against emerging threats.

Stay proactive. Stay secure.

About QuillAudits

QuillAudits is a leading name in Web3 security, offering top-notch solutions to safeguard projects across DeFi, GameFi, NFT gaming, and all blockchain layers. With seven years of expertise, we've secured over 1400 projects globally, averting over $3 billion in losses. Our specialists rigorously audit smart contracts and ensure DApp safety on major platforms like Ethereum, BSC, Arbitrum, Algorand, Tron, Polygon, Polkadot, Fantom, NEAR, Solana, and others, guaranteeing your project's security with cutting-edge practices.

7+ Years of Expertise	**1M+** Lines of Code Audited
50+ Chains Supported	**1400+** Projects Secured

Follow Our Journey

AUDIT REPORT

October 2025

For

omnyx

QuillAudits

Canada, India, Singapore, UAE, UK

www.quillaudits.com audits@quillaudits.com