coinspect

You build, we defend.

Source Code Audit

Firelight dApp

September 2025

Firelight Interface

Source Code Audit

Version: v250919 | Prepared for: Sentora | September 2025

Security Assessment

1. Executive Summary

In **September 2025**, **Sentora** engaged Coinspect to perform a security audit of the Firelight dApp. The objective of the project was to evaluate the security of the application's client-side interface and its integration with the Flare Network.

The Firelight project is a liquid staking protocol built on the Flare Network. It provides a decentralized application (dApp) frontend that allows users to stake FXRP in curated vaults. In exchange for their deposits, users receive stXRP, a Liquid Staking Token (LST) representing their share in the vault. The initial launch is designed to feature a single, secure vault with no slashing risks to encourage early adoption.

2. Summary of Findings

This section provides a concise overview of all the findings in the report grouped by remediation status and sorted by estimated total risk.

2.2 Findings where caution is advised

Issues with risk in this list have been addressed to some extent but not fully mitigated. Any future changes to the codebase should be carefully evaluated to avoid exacerbating these issues or increasing their probability.

Findings with a risk of None pose no threat, but document an implicit assumption which must be taken into account. Once acknowledged, these are considered solved.

Id	Title	Risk
FIRE-005	Insufficient testing	Low

2.3 Solved issues & recommendations

These issues have been fully fixed or represent recommendations that could improve the long-term security posture of the project.

Id	Title	Risk
FIRE-001	Supply chain vulnerability in MetaMask SDK	High
FIRE-002	Missing chain segregation leads to inaccurate cached data	Low
FIRE-003	Missing Content Security Policy (CSP)	Low
FIRE-004	Lack of Subresource Integrity (SRI)	Low

FIRE-006	Stale analytics data due to missing cache invalidation	None
FIRE-007	Obsolete code artifacts increases codebase complexity and maintenance	None
FIRE-008	Lack of geo blocking leaves the dApp accessible from prohibited regions	None
FIRE-009	Use of deprecate clipboard API creates compatibility risk	None

3. Scope

The scope was set to be the repository at `https://github.com/firelight-protocol/interface` at commit `955c49b978b5711aac465f04dea270120f215c04`.

The following components were not included in this review's scope:

- The Firelight backend API.
- The Firelight smart contracts.

4. Assessment

The Firelight frontend is a decentralized application developed with React for production and using the Vite tool as a development server. This dApp is designed to provide a user interface for vault staking on the Flare Network. The application is built with a technical stack, including Vite, Wagmi for blockchain interactions, RainbowKit for wallet connectivity, and TanStack Query for state management and caching. Its architecture is component-based, centered around a "Wizard" pattern for guiding users through multi-step transactions like deposits and withdrawals.

The application demonstrates an adherence to many web3 security best practices. Key positive aspects include:

- **Transaction Lifecycle**: The use of Wagmi's `useWaitForTransactionReceipt` hook ensures that the UI updates and subsequent actions only occur after a transaction has been successfully confirmed on-chain. The application provides clear, real-time feedback to the user during pending, successful, and failed transaction states.
- **Secure Token Approvals**: The `DepositWizard` provides an exemplary implementation of the ERC20 approval flow. It does not request infinite approval by default and instead offers it as a clear, user-controlled opt-in via a checkbox, mitigating the risks associated with unlimited token allowances.

4.1 Security assumptions

This security assessment was conducted with the following assumptions:

- The backend API and the Firelight smart contracts are assumed to be secure, audited, and non-malicious. The data they provide is correct, but not necessarily safe for direct rendering in the frontend without validation.
- The smart contract ABIs used within the frontend codebase are accurate and perfectly match the deployed, audited contracts.
- The frontend application relies on the backend API as the authoritative source for smart contract addresses used in transaction construction. This assessment assumes that the API will always provide the correct and legitimate contract addresses for all on-chain operations.

4.2 Decentralization

The Firelight frontend operates as a client-side interface to on-chain smart contracts, but it introduces several vectors of centralization:

- **Backend API Dependency**: The application depends on a centralized backend API for fetching all supplementary data, including vault lists, user positions, and historical transactions. The availability and integrity of the dApp are therefore directly tied to the availability and security of this API. If the API were compromised, an attacker could provide a malicious `vault_address`, tricking the user into signing a transaction that interacts with an attacker's contract instead of the legitimate Firelight vault.

4.3 Testing

As highlighted in `FIRE-005`, the Firelight testing process lacks a formal, automated test suite to validate core business logic and functionality. The project relies solely on Storybook for visual UI verification. Furthermore, these Storybook tests are populated with inaccurate and unrealistic mock data, which undermines their reliability for catching bugs.

4.4 Code quality

The codebase is well-structured and meets the requirements for the initial single-vault launch. To ensure long-term stability and reliability, key areas for improvement include updating dependencies to remediate a supply chain vulnerability and replace a deprecated browser function. Additionally, the caching logic should be refined to ensure data integrity when users switch between networks and to provide immediate UI feedback after transactions are completed.

5. Detailed Findings

FIRE-001

Supply chain vulnerability in MetaMask SDK

Status
Solved

Resolution
Fixed

Risk
High

Impact
High

Likelihood
High

Location

```
package-lock.json
```

Description

The application is exposed to a supply chain attack. A production dependency, `@metamask/sdk`, includes a compromised version of the debug package `debug@4.4.2`.

According to security reports , the malicious code in `debug@4.4.2` is designed to:

- Hijack transactions: intercept calls to the user's wallet and silently redirect funds to an attacker's address.
- Manipulate data: alter network responses to deceive the user and the application.

This is not a flaw in Firelight's code, but in a third-party dependency that was maliciously altered.

Recommendation

Upgrade to a non-compromised version of the `wagmi` library.

Status

Fixed on commit `6b79de81cf6ce4c410ca94ef20a7cdb67514a3a1`.

The Firelight team's updates have resulted in a dependency tree where the malicious `debug@4.4.2` package is no longer present. Although `npm audit` still reports the vulnerability, it has been resolved through npm's dependency deduplication, which now points to a safe version of the `debug` package in the current build. However, installation in a different environment could potentially re-introduce the vulnerable version. To ensure long-term fix, Coinspect provides the following follow-up recommendations:

- Consider adding an overrides block to `package.json` to enforce the secure version of `@metamask/sdk` across the entire dependency tree.
- Upgrade the `wagmi` package to a newer version.
- Apply `npm audit fix` and verify with `npm audit` that no vulnerabilities were left pending.

FIRE-002

Missing chain segregation leads to inaccurate cached data

Status
Solved

Resolution
Fixed

Risk
Low

Impact
Low

Likelihood
Medium

Location

```
src/hooks/api/useDeposits.ts
```

Description

The Firelight data cache is not segregated by blockchain network. When a user switches networks (e.g., from Flare Testnet Coston2 to Flare Mainnet), the application incorrectly displays cached data from the previous network.

The root cause is that the cache keys for data queries, such as in the `useDeposits` hook, are based solely on the user's address and are missing the `chainId`. This lead inaccurate data on mainnet.

Additionally, the application lacks error handling for network requests. This causes UI components, such as the deposit and withdraw input fields, to fail silently and go blank when an underlying RPC request fails on the mainnet.

Recommendation

The application's data fetching and caching logic must be made network-aware to ensure data integrity.

Status

Fixed on commits `5263a3a9ffa556183ffe9dac921de03812591300` and `035f9ad3ae4ab8beb7ae93e12393186b0d3ca181`.

The Firelight team has addressed both aspects of this finding. The `chainId` has been added to all relevant query keys, ensuring that cached data is correctly segregated by network. Additionally, error handling has been implemented for the API requests.

FIRE-003

Missing Content Security Policy (CSP)

Status
Solved

Resolution
Fixed

Risk
Low

Impact
Medium

Likelihood
Low

Location

`index.html`

Description

The application lacks a Content Security Policy (CSP), a critical security layer that helps mitigate a variety of attacks. The absence of a CSP header exposes the application to risks such as clickjacking and data exfiltration.

Specifically, the following vulnerabilities exist:

- **Clickjacking:** Without a `frame-ancestors 'none'` or `frame-ancestors 'self'` directive, a malicious actor can embed the application in an iframe on a hostile website. This can be used to trick users into performing unintended actions on their accounts by overlaying invisible UI elements.
- **Data Exfiltration & Unauthorized Resource Loading:** The lack of directives like `script-src`, `style-src`, `connect-src`, and `img-src` allows the browser to load and execute resources from any origin. If an attacker manages to inject even a small piece of malicious code (e.g., through a compromised

dependency), they could load external scripts to steal sensitive user data or deface the application by loading unauthorized images and styles.

Recommendation

Implement a strict Content Security Policy (CSP).

Status

Fixed.

The Firelight team has stated that a Content Security Policy is enforced at the infrastructure level via their CDN configuration.

FIRE-004

Lack of Subresource Integrity (SRI)

Status **Solved**	Risk **Low**
✓	🟥⬜⬜
Resolution **Fixed**	Impact **Low** Likelihood **Low**

Location

```
index.html
```

Description

The application loads external resources, such as Google Fonts, without using Subresource Integrity (SRI). An adversary can abuse this problem to inject malicious code if the external resource is compromised.

SRI is a security feature that enables browsers to verify that resources they fetch (for example, from a CDN) are delivered without unexpected manipulation.

Recommendation

Use Subresource Integrity (SRI) for all external resources.

Status

Fixed on commit `980d76f76a6829776b7948875851d4fe0a62b96f`.

The risk was mitigated by removing the external dependency on Google Fonts and self-hosting the required font files directly. This eliminates the need for Subresource Integrity (SRI) for this asset, as it is no longer fetched from a third-party domain.

FIRE-005

Insufficient testing

Status
Caution Advised

Resolution
Acknowledged

Risk
Low

Impact
Low

Likelihood
Low

Location

```
src/mock/withdrawals.ts
src/components/ui/Address/index.stories.tsx
```

Description

The Firelight testing process is marked by the absence of a thorough test suite. This means that core business logic and utility functions are not systematically validated to ensure that new code changes do not inadvertently break existing functionality. While the project uses Storybook for visual UI testing, this does not compensate for the lack of automated logic validation, creating a risk of introducing functional bugs during future development cycles.

This gap is compounded by the use of inaccurate mock data. By testing components against unrealistic states that do not reflect real-world scenarios, the process can mask bugs related to data validation and transaction construction. For example, some Storybook components rely on single, static address values:

```
export const Default: Story = {
  render: Template,
  args: {
    address: "0x2260fac5e5542a773aa44fbcfedf7c193bc2c599",
  },
};
```

In the other cases, protocol's mock data files use the zero address for fields like `vault_address`. Since the protocol would never permit interactions with the zero address, components are being tested against a state that cannot exist in production. This practice can mask bugs related to data validation, rendering, and transaction construction.

```
// Using the zero address for mock data is unrealistic and can hide
bugs.
export const MOCK_WITHDRAWALS = [
  {
  // ...
    vault_name: "stFXRP",
    vault_address: "0x0000000000000000000000000000000000000000",
  // ...
    },
  },
  // ...
];
```

Recommendation

Implement a thorough testing suite that includes unit and integration tests for business logic, hooks, and utility functions.

Replace hardcoded and invalid addresses in all test and mock data with a diverse and realistic set of valid, checksummed addresses.

Status

Acknowledged.

The Firelight team has acknowledged the findings regarding the lack of test suite and the use of unrealistic mock data.

FIRE-006

Stale analytics data due to missing cache invalidation

Status

Solved

✓

Resolution

Acknowledged

Risk

None

Impact

Recommendation

Likelihood

-

Location

```
src/components/DepositWizard/steps/Stake.tsx
src/components/WithdrawalWizard/steps/Withdraw.tsx
```

Description

Global analytics charts, such as "Total Inflows," do not update immediately after a user completes a `deposit` or `withdrawal`. This is caused by a failure to invalidate the global analytics cache after a transaction succeeds.

The transaction success logic in components like `DepositWizard` correctly invalidates user-specific data caches but omits the invalidation call for the global analytics cache, which uses the ['analytics'] query key.

This results in a delay of up to 60 seconds before the UI reflects the user's action, degrading the user experience.

Recommendation

Consider updating the transaction success handlers in both the `DepositWizard` and `WithdrawalWizard` to also invalidate the global analytics cache.

Status

Acknowledged.

The Firelight team has acknowledged that the global analytics cache is not invalidated immediately after user transactions, which can lead to a temporarily stale UI.

FIRE-007

Obsolete code artifacts increases codebase complexity and maintenance

Status
Solved

✓

Resolution
Fixed

Risk
None

Impact
Recommendation

Likelihood
-

Location

```
src/sample_data/
src/utils/blockchain.ts
src/components/ui/ChartCard/index.tsx
src/components/ui/Table/index.stories.tsx
```

Description

The codebase contains several instances of obsolete, commented-out, and unused code, which increases maintenance overhead. While not a direct security threat, these artifacts degrade the overall quality.

The review identified several instances of these obsolete artifacts:

- Unused sample data: The `sample_data` directory contains mock data related to traditional stocks information, which is irrelevant to the protocol's service.

- No accurate default logic: The `ChartCard` component includes a fallback to a `stockChart` constructor, which is a remnant of placeholder code and does not align with the application's purpose.

```
export const Chart = forwardRef((props: any, ref:
React.Ref<HighchartsReactRefObject> | undefined) => {
  return (
    <Container>
      <HighchartsReact highcharts={Highcharts} constructorType=
{props.constructorType ?? "stockChart"} options={props.options} ref=
{ref} />
    </Container>
  );
});
```

- Obsolete file: The file `blockchain.ts` is entirely commented out, confirming it is no longer in use and should be removed.
- Irrelevant storybook data: The Storybook file for the generic Table component is populated with a large, hardcoded list of rock band members.

Coinspect considers these artifacts do not introduce a vulnerability but rather degrade code quality and increase the risk of future development errors.

Recommendation

Conduct a thorough code cleanup to identify and remove all obsolete, unused, and commented out artifacts from the codebase.

Status

Fixed on commit `e2afe134163ac652aea433441999b4c300e0c8dd`

The team has addressed most of the identified issues by removing the `sample_data` and `blockchain.ts` files and updating the Storybook data to be relevant to the application. However, the `ChartCard` component's default fallback to `stockChart` remains, which the team has stated is the intended behavior.

FIRE-008

Lack of geo blocking leaves the dApp accessible from prohibited regions

Status
Solved

✓

Resolution
Fixed

Risk
None

Impact
Recommendation

Likelihood
-

Location

`src/components/TermsModal/index.tsx`

Description

The application `Terms of Use` mention jurisdictional restrictions, but no technical mechanism is implemented to enforce them, failing to block users from prohibited regions. This exposes the Firelight project to legal and regulatory risk by creating a false sense of compliance while the protocol remains openly accessible to users from sanctioned jurisdictions.

The Terms of Use explicitly state that users from certain sanctioned countries or territories are prohibited from using the service. However, the user appearing to be from a restricted region could still access and interact with the full functionality of the application without any hindrance.

The root cause is the absence of any geo-blocking logic within the `TermsModal` component. The code does not contain any logic to perform a client-side check of the user's location.

Recommendation

Consider implementing a mechanism to correctly enforce the protocol's jurisdictional restrictions.

Status

Fixed.

The Firelight team has confirmed that geo-blocking is implemented at the infrastructure level, which is outside the scope of this code audit.

FIRE-009

Use of deprecate clipboard API creates compatibility risk

Status

Solved

Resolution

Fixed

Risk

None

Impact

Recommendation

Likelihood

-

Location

`src/utils/common.ts`

Description

The Firelight dApp "copy address" feature relies on the `document.execCommand('copy')` browser API. This API is officially deprecated and no longer recommended by web standards.

```
export const copy_to_clipboard = async (text: string) => {
  const textArea = document.createElement("textarea");
  textArea.style.top = "0";
  textArea.style.left = "0";
  textArea.style.position = "fixed";
  document.body.appendChild(textArea);
  textArea.value = text;
  textArea.focus();
  textArea.select();
  document.execCommand('copy'); // Deprecated API call
```

```
    document.body.removeChild(textArea);
  }
```

According to Mozilla documentation, this feature is kept only for compatibility purposes and "may cease to work at any time."

Continuing to use this obsolete API creates a forward-compatibility risk. A routine browser update could cause this core user experience feature to fail unexpectedly for all users, degrading the application's quality and reliability.

Recommendation

Refactor the address copying functionality to use the modern, asynchronous Clipboard API .

Status

Fixed on commit `4b8b60eb84787959a483709e8bb29002e742585f`

The deprecated `document.execCommand('copy')` API has been replaced with the asynchronous Clipboard API `navigator.clipboard.writeText`. This change resolves the forward-compatibility risk.

6. Disclaimer

The contents of this report are provided "as is" without warranty of any kind. Coinspect is not responsible for any consequences of using the information contained herein.

This report represents a point-in-time and time-boxed evaluation conducted within a specific timeframe and scope agreed upon with the client. The assessment's findings and recommendations are based on the information, source code, and systems access provided by the client during the review period.

The assessment's findings should not be considered an exhaustive list of all potential security issues. This report does not cover out-of-scope components that may interact with the analyzed system, nor does it assess the operational security of the organization that developed and deployed the system.

This report does not imply ongoing security monitoring or guaranteeing the current security status of the assessed system. Due to the dynamic nature of information security threats, new vulnerabilities may emerge after the assessment period.

This report should not be considered an endorsement or disapproval of any project or team. It does not provide investment advice and should not be used to make investment decisions.