Security Audit

August - September, 2024

Sky.money: Phase Zero, Phase One, and Helipad

Audited by
Ethan Bennett

Contents

About Darklinear Solutions

Darklinear Solutions provides unrivaled security for blockchain applications, from the bytecode to the browser. With years of experience in smart contract development and traditional software engineering, we find the bugs that others miss. Learn more at darklinear.com.

Introduction

As part of MakerDAO's endgame roadmap and its rebrand to Sky, one of the Ecosystem Actor teams (Jetstream) built opensource code that can be used to create frontend apps that interact with the new contracts. This audit and penetration test covers these codebases, including a webapp that was built from them (app.sky.money) and a smaller app that preceded its launch. More details about the scope of the audit can be found in the Summary of Engagement section.

Disclaimer: This audit is not a guarantee that the reviewed code can never be exploited, and any further development will require further review.

Risk Classification

The issues uncovered during the course of this review can each be described by one of the following three identifiers:

- **Medium risk** findings represent vulnerabilities that could significantly impact user security or proper functionality of the application. These issues might allow attackers to compromise user data, manipulate application behavior, or exploit security controls, but typically require specific conditions or face practical exploitation barriers.
- **Low risk** findings represent security weaknesses that could be exploited under limited circumstances or would have minimal impact if exploited. These might stem from misconfiguration, deviations from security best practices, or edge cases in security controls. While exploitation is possible, it often requires additional vulnerabilities or specific user actions to be impactful.
- **Informational** findings represent opportunities to improve security posture, development practices, or user experience without addressing an immediate security risk. These might include outdated dependencies, suboptimal configurations, or areas where defense-in-depth could be enhanced.

Urgency according to risk classification

Risk level	Recommendation
High	Must fix
Medium	Should fix
Low/Informational	Could fix

Summary of Engagement

Jetstream requested a security audit of three codebases: Phase Zero App, Phase One App, and Helipad. A different Ecosystem Actor team used Phase One App to create app.sky.money, and Phase Zero App was deployed to the same subdomain with some limited features made available prior to the larger app's launch. Much of the core logic for both apps is implemented in the hooks and widgets in the Helipad repository.

The main body of work in this audit began on August 7, 2024, and concluded with the launch of Phase One App on September 18, 2024. It consisted of four main components:

- Breadth-focused manual review of all three codebases, focused on threat modeling and mapping the flow of data through both apps
- Depth-focused manual review of higher-risk functionality identified in breadth-focused review, including:
  - Sky smart contract interactions
  - CoWSwap interactions
  - Requests and responses to/from the backend
  - Wallet authentication
  - Signed message validation
  - Arbitrary data rendering (cross-site scripting risks)
  - Restrictions for blocked regions, sanctioned addresses, and VPN users
- Static analysis using Semgrep and CodeQL (more details are specified in Static Analysis Configuration)
- Penetration testing using Burp Suite Pro (more details are specified in Penetration Testing Configuration)

Due to active development of all three codebases in parallel to this audit, a number of commits were reviewed. The final reviewed commit for Helipad is listed below. The Jetstream team cleared the commit history for both Phase One App and Phase Zero App shortly after the conclusion of this audit, so the squashed initial commits for each are referenced below (and wherever code is linked throughout this report). There were no major changes between the final reviewed commits in the audit scope for either repo and the squashed commits referenced below.

- Phase Zero App: #0e0ad4a
- Phase One App: #8e63ffa
- Helipad: #e0773f8

Summary of Findings

The descriptions for some findings reveal private information about the teams or infrastructure involved. These findings have been removed from this public version of the report.

Classification	Finding
Medium risk (M-01)	Potential for stored XSS via unsanitized `termsLink`
Medium risk (M-02)	Insecure ETH address storage is compounded by VPN policy
Medium risk (M-03)	Resolved and removed from public report
Medium risk (M-04)	Resolved and removed from public report
Low risk (L-01)	Data from subgraphs lacks explicit validation
Low risk (L-02)	HSTS `maxAge` is too low
Low risk (L-03)	Content Security Policy should restrict `frame-ancestors` to allowed origins
Low risk (L-04)	Should use Permissions Policy to reduce attack surface
Informational (I-01)	Most token approval transactions could be eliminated by using `permit` with a relayer
Informational (I-02)	Supply chain could be better secured using LavaMoat
Informational (I-03)	Static analysis could be added to Github Actions
Informational (I-04)	Outdated and vulnerable dependencies
Informational (I-05)	Acknowledged and removed from public report
Informational (I-06)	`usePriceImpact` never returns `cowPricesError`
Informational (I-07)	Unreadable text on Phase One App's 404 page

Findings

M-01: Potential for stored XSS via unsanitized `termsLink`

Severity: Medium risk

Context: UnauthorizedPage.tsx, ExternalLinkModal.tsx (Phase One App)

Description: The Terms and Conditions of using Sky (which are initially signed as a cryptographic message when users first connect to the app) are available on a separate subdomain. This external version of the Terms is linked in lieu of displaying their full text in-app, except for when the user initially connects their wallet.

The specification of these terms is a process designed to be performed by anyone who forks the codebase to create their own frontend. For this reason, the url is defined as an environment variable, which itself is accessed anywhere the link needs to be inserted into the page. The environment variable, which is stringified JSON (with a `name` and a `url`) is passed through `JSON.parse`, and then `termsLink[0].url` is immediately and directly inserted as the `href` for an `ExternalLink`:

```
<ExternalLink
  skipConfirm
  className="text-textEmphasis"
  showIcon={false}
  href={termsLink[0].url}
>
  {termsLink[0].name}
</ExternalLink>
```

`href` is one of the only contexts in which React will not automatically sanitize strings, so an attacker could inject malicious code by way of this `termsLink` — for example, they could update the environment variable to `VITE_TERMS_LINK={"name": "maliciousLink", "url": "javascript:alert('XSS')"}`. While access to update the environment variable in Phase One App is limited, this vulnerability would allow any attacker who, for example, manages the relatively less sophisticated challenge of compromising a team member's account with the means to execute a much more potent attack on all of Sky's users.

Recommendation: This issue can be avoided by validating the protocol of the provided url before passing it into the `href`.

Below is an example of how an instance of the `termsLink` could be updated to mitigate this issue. For the original implementation, see `UnauthorizedPage.tsx#L83`.

```
let termsLink: any[] = [];
try {
  const linkJSON = JSON.parse(import.meta.env.VITE_TERMS_LINK);
  const parsedUrl = new URL(linkJSON.url);
  // Ensure that the url begins with 'https:'
  if (parsedUrl.protocol === 'https:') {
    // Set the terms link
    termsLink = linkJSON;
  } else {
    // Validation failed; don't use the environment variable
    termsLink = null;
  }
} catch (error) {
  console.error('Error parsing terms link');
}
```

Resolution:

- Jetstream: Fixed in PR #5.
- Darklinear: Confirmed.

M-02: Insecure ETH address storage is compounded by VPN policy

Severity: Medium risk

Context: Phase One App, Phase Zero App

Description: Fingerprinting scripts across the internet can create a unique ID for anyone by combining arbitrary details of their browsing environment like the screen size, the installed fonts, and the browser's User Agent. This is the basis of modern tracking in the browser, and it can be difficult for privacy-oriented users to combat. Additionally, `wagmi` stores data about the user's current session in `localStorage`, which is not secure.

This combination of circumstances creates the opportunity for malicious actors to access `localStorage` and associate a user's Ethereum address with their browsing fingerprint.

While an attacker would need to find a way to access `localStorage`, this is relatively easy to accomplish. And it's particularly easy if the malicious actors are not attackers per se, but are otherwise legitimate software providers with invasive policies regarding user data. There is no way for a user to prevent their address from being exposed to all activated browser extensions (not to mention to the browser itself).

Recommendation: Options are limited: regulatory compliance is not optional, there is currently no wallet provider that would store the user's address more securely, and encrypting this data in `localStorage` risks breaking the wallet's functionality. And, assuming that there is no way to sufficiently comply with regulations without fully blocking some regions and users, it is difficult to conceive of a way to satisfy the app's legal requirements without also disabling VPNs.

Given these significant limitations, the best option in the short term is to warn users that their wallet address will be exposed to any browser extensions that are active in the user's session. Since this concern

is likely to be more or less limited to users who attempt to access the app with a VPN in the first place, it would be reasonable to include this warning in the same message that instructs them to disable their VPN and reload the page. This could be phrased in such a way that it does not discourage the user from using the app, but does encourage security (e.g., "be sure that you trust any browser extensions you have enabled before refreshing the page").

For better coverage, this warning could also be added to the `TermsModal`.

Resolution:

- Jetstream: We've passed this to the marketing team to decide what to display for a warning.

M-03: Resolved and removed from public report

Severity: Medium risk

Description: An adequate description of this finding would reveal private information, so it has been removed from the public version of this report.

M-04: Resolved and removed from public report

Severity: Medium risk

Description: An adequate description of this finding would reveal private information, so it has been removed from the public version of this report.

L-01: Data from subgraphs lacks explicit validation

Severity: Low risk

Context: Helipad (`hooks`)

Description: Throughout Helipad's `hooks` package, the app fetches data from subgraphs. This data is used across the app to determine whether certain actions are available to users, to render information about users' account balances, to populate charts, and for other reasons.

Although it would be difficult to manipulate, this data still ultimately comes from a source outside the Jetstream team's control. It then passes through a multi-stage mechanism for indexing and serving the data, which itself is only partially within the control of Jetstream, before it finds its way into Helipad or Sky's UI components.

This data is never injected directly into HTML or JavaScript templates, so the worst potential impacts of its manipulation are not relevant. Nonetheless, ensuring that this data conforms to some basic requirements can prevent more severe vulnerabilities from emerging here in the future.

Recommendation: These queries validate subgraph data implicitly when they transform the raw responses into objects. They should also implement some level of explicit validation by filtering any obviously incorrect values that would slip past these implicit type checks undetected. For example, if

the all-time total amount supplied to the Savings contract comes back as 0, the hook should throw an error. This requires individual analysis of each hook and the data it is responsible for fetching. The goal in each case is to catch any unrealistic or impossible values before they return to Phase One App.

Resolution: Acknowledged.

L-02: HSTS `maxAge` is too low

Severity: Low risk

Context: Phase One App, Phase Zero App

Description: HTTP Strict Transport Security (HSTS) ensures that browsers can only connect to an app via HTTPS. This still leaves users vulnerable to man-in-the-middle attacks on their first connection, however, so the HSTS header has a `preload` option. `preload` adds the domain to a list in the browser, and the browser will reference this list and upgrade the first request if a user mistakenly attempts an HTTP request.

Since this header will also block HTTP requests within the app, the `preload` header could render an app completely unusable. To help prevent this, it also offers a `max-age` parameter: this is the expiration date of the HSTS header, and it's generally recommended to be set to one year. In both Phase One App and Phase Zero App, the `max-age` is set to thirty days.

Recommendation: The `Strict-Transport-Security` header's `max-age` should be boosted to `31536000`.

Resolution: Acknowledged.

L-03: Content Security Policy should restrict `frame-ancestors` to allowed origins

Severity: Low risk

Context: vite.config.ts (Phase One App)

Description: Phase One App allows itself to be framed so it can work in the Safe app, which renders all apps in iframes by default. However, this can be controlled more granularly in the Content Security Policy by way of the `frame-ancestors` directive.

Allowing the app to be rendered in an iframe by any host creates an opportunity for clickjacking attacks. While these have some built-in mitigation in the context of blockchain apps (since any state-changing operation would require the user's signature), they could still fool users into signing a transaction they don't intend to sign.

Recommendation: The team should identify all the hosts that should be able to frame Phase One App and specify them explicitly as the allowed `frame-ancestors` in the Content Security Policy. Note that mobile wallets may also implement their in-app browsers this way, so be sure to test this in a staging environment before adding it to the production app.

Resolution: Acknowledged.

L-04: Should use Permissions Policy to reduce attack surface

Severity: Low risk

Context: Phase One App, Phase Zero App

Description: Modern browsers have a wide range of features that can be enabled to access sensitive data from the user's device. These allow apps to enable the user's camera or microphone, read their geolocation, calculate their current walking speed, and access all kinds of other data that can be used to identify the user or deduce things about their behavior.

Although none of these features are accessed or directly exploitable in either Sky app, they can be explicitly disabled. This greatly limits what an attacker could do even if an injection vulnerability emerges in the future.

Recommendation: Set a Permissions Policy header to disable access to any information from the user's device that the app does not need. Here is an example of a restrictive Permissions Policy implemented as a `meta` tag:

```
// vite.config.ts: parse and inject like CSP
const PERMISSIONS_POLICY = `
  accelerometer=(),
  ambient-light-sensor=(),
  battery=(),
  camera=(),
  display-capture=(),
  document-domain=(),
  encrypted-media=(),
  geolocation=(),
  gyroscope=(),
  magnetometer=(),
  microphone=(),
  midi=(),
  payment=(),
  picture-in-picture=(),
  publickey-credentials-get=(),
  screen-wake-lock=(),
  usb=(),
  web-share=(),
  xr-spatial-tracking=(),
  clipboard-read=(),
  clipboard-write=(),
  gamepad=(),
  hid=(),
  idle-detection=(),
  serial=(),
`
```

Resolution: Acknowledged.

I-01: Most token approval transactions could be eliminated by using `permit` with a relayer

Severity: Informational

Context: Phase One App, Helipad

Description: Although it isn't part of the ERC-20 standard, most ERC-20 tokens since the adoption of ERC-2612 include a `permit` function. `permit` allows users to set approvals without sending an additional transaction by instead signing a message, which itself can be submitted by a third-party relayer contract and executed atomically with the user's main transaction.

Recommendation: Jetstream should consider developing relayer smart contracts, which would greatly improve the app's UX by eliminating the need for most `approval` transactions. This would need to account for tokens that do not implement a `permit` function, and it should limit this functionality to EOAs.

Uniswap's Universal Router is a great example to reference. But the Sky relayer would be much simpler: from its finite list of supported tokens, only MKR and USDT pre-date ERC-2612 (and are not upgradeable).

Resolution: Acknowledged.

I-02: Supply chain could be better secured using LavaMoat

Severity: Informational

Context: Phase One App, Helipad

Description: One of the more difficult areas to secure in any codebase is its supply chain. If any of an app's dependencies is malicious or exploited, attackers can inject arbitrary code where it is least likely to be discovered.

Recommendation: The most common way attackers inject malicious code into vulnerable dependencies is by way of malicious lifecycle scripts (e.g. `postinstall`). LavaMoat's `allow-scripts` can block dependencies from executing these scripts when they are installed, and it also enables explicit allow-listing for any scripts that need to run for legitimate reasons. Any script that it blocks will throw an error, alerting the team to a potentially malicious dependency that needs to be investigated further.

Note that the decision to allow any necessary lifecycle scripts still requires manual evaluation, so this tool will not eliminate the need for careful dependency management.

Resolution: Acknowledged.

I-03: Static analysis could be added to Github Actions

Severity: Informational

Context: Phase One App, Helipad

Description: Static analysis of code can automatically detect a wide range of common vulnerabilities. This can be added to Github Actions to reduce the likelihood of vulnerabilities being merged into production undetected.

Recommendation: Both Semgrep and CodeQL are great options, and both allow for the development of custom rules to detect in analyses. However, Semgrep is easier to set up and work with, so it would be the better first step.

Semgrep is powerful with its default settings alone, but when integrated with CI, it's best to develop an intentional policy for it. Some issues can be set to block the build, for instance, while others can be set to report only. And while it can uncover high severity issues, it will also report false positives, so using the tool is not an entirely automatic process.

Resolution: Acknowledged.

I-04: Outdated and vulnerable dependencies

Severity: Informational

Context: Phase One App, Helipad, Phase Zero App

Description: Although none of them are directly exploitable in any of the audited code, all three codebases use several outdated dependencies with known vulnerabilities.

Recommendation: The team already uses Dependabot, which is a good start. These findings can be supplemented by Semgrep's dependency analysis. Both should be integrated with the development process on a regular basis.

Additionally, dependencies should be kept up-to-date with their most recent major version, since bug fixes and vulnerability patches may have occurred even if they weren't flagged by automated dependency scanners.

For a full list of outdated dependencies in any project, run `pnpm outdated`.

Resolution: Acknowledged.

I-05: Acknowledged and removed from public report

Severity: Informational

Description: An adequate description of this finding would reveal private information, so it has been removed from the public version of this report.

I-06: `usePriceImpact` never returns `cowPricesError`

Severity: Informational

Context: `usePriceImpact.tsx` (Helipad)

Description: `usePriceImpact` is a Helipad hook responsible for estimating the real price of an asset in a prospective CoWSwap trade, relative to that asset's price in the broader market. In order to do this, the hook first fetches the spot price of the token from the Block Analytica API, then it fetches the actual price estimate for the user's swap from CoWSwap's API.

In the event of an error from either request, the hook attempts to dynamically define the `error` variable (which is subsequently returned by the hook):

```
const error =
  baPricesError && cowPricesError ? baPricesError
    : baPricesError ? null
    : cowPricesError ? null
    : null;
```

This statement will either return `baPricesError` or `null`, but never `cowPricesError`.

Recommendation: Simplify this statement and ensure that it returns the intended error in each condition.

Resolution: Acknowledged.

I-07: Unreadable text on Phase One App's 404 page

Severity: Informational

Context: NotFound.tsx (Phase One App)

Description: When a user attempts to navigate to a non-existent page, the "Not Found" page displays black text against the Sky background (with no modal). This is difficult to read, and it is the only instance of text displayed directly on the background besides the footer. This suggests that the styling on this page is incomplete.

Recommendation: Make this message easier to read by changing its color to white and putting it in a modal.

Resolution: Acknowledged.

Static Analysis Configuration

Semgrep is easier to integrate and use, but CodeQL can complement it with deeper and more configurable scans. When using either or both, false positives are common, so findings from static analysis are more often starting points for further investigation than they are fully fledged vulnerabilities.

Semgrep

Semgrep is a lighter-weight pattern matching tool that can detect vulnerabilities in code (including dependencies). A wide range of first-party and community rulesets can be added in the Semgrep UI, and each ruleset can be configured to optionally block the build if they find an issue (when integrated with Github Actions).

In this audit, Semgrep was run using `semgrep ci` with the following rulesets applied:

- default
- gitleaks
- secrets
- owasp-top-ten
- react
- react-best-practices
- typescript
- r2c-security-audit
- security-audit
- xss

The results of these runs will be logged in Semgrep's UI, which comes with some quality-of-life features for triaging real issues and hiding false positives from view.

CodeQL

CodeQL (acquired by GitHub) creates a queryable database to represent the target code (its structure, data flow, control flow, etc.). Users can then write and run queries against the database to search for anything that can be expressed in its SQL-like syntax.

In this audit, CodeQL was run with the following two commands:

- `codeql database create codeql.db --language=javascript`
- `codeql database analyze codeql.db --format=sarif-latest --output=results.sarif -- codeql/javascript-queries`

The second command runs CodeQL's first-party JavaScript queries against the database created from the code, and outputs the results in the SARIF format. These results can be easily navigated using the SARIF Explorer VSCode extension.

Penetration Testing Configuration

This audit utilized Burp Suite Pro in four distinct ways:

- Live scanning during manual webapp interaction
  - Light-weight background scanning during normal usage of the app
- Active scanning
  - Invasive automated testing
- DOM Invader
  - Scanning for DOM XSS vulnerabilities during normal usage of the app
  - Scanning for prototype pollution vulnerabilities during normal usage of the app
- Burp Repeater
  - Used to repeat and manipulate specific requests and responses

In addition to the above strategies, a variety of Burp extensions were added to extend the functionality of its scans:

- Param Miner
  - Detects hidden and unkeyed headers and parameters, which can be used to identify cache poisoning vulnerabilities, timing attacks, and fat GET requests, as well as many less severe issues
- Active Scan++
  - Significantly extends active and passive scanning capabilities
- Backslash Powered Scanner
  - Extends the server-side injection capabilities of Burp's scans
- Additional Scanner Checks
  - Checks for vulnerabilities not covered in the default scans
- HTTP Request Smuggler
  - Attempts to send request smuggling payloads
- Software Vulnerability Scanner
  - Scans for known vulnerabilities