Drift Protocol

Audit

Presented by:

OtterSec contact@osec.io

Caue Obici caue@osec.io

Robert Chen r@osec.io

Contents

Appendices

01 | Executive Summary

Overview

Drift Labs engaged OtterSec to perform an assessment of the `snap-solana` program. This assessment was conducted between August 2nd and August 4th, 2023. For more information on our auditing methodology, see Appendix B.

Key Findings

Over the course of this audit engagement, we produced 3 suggestions in total.

We provided recommendations concerning the snap dialog's display of specific crucial transaction details to the user before approval (OS-DFT-SUG-00), which could lead to financial losses to a user who would blindly approve a transaction. Additionally, we suggested addressing unpinned npm dependencies, which may render the application susceptible to supply chain attacks (OS-DFT-SUG-01). These vulnerabilities can have severe consequences since they would alter the function of the snap, which has privileged permissions.

We also recommended remediations against type confusion issues within the RPC parameters (OS-DFT-SUG-02). Although we did not find any vulnerabilities that are related, it may prevent future attacks against potential vulnerabilities introduced in new features.

02 | Scope

The source code was delivered to us in a git repository at github.com/drift-labs/snap-solana. This audit was performed against commit 04fa431.

A brief description of the programs is as follows.

Name	Description
snap-solana	A Metamask Snap that facilitates the connection between Metamask and Drift, enabling Metamask to function as a Solana wallet. This snap empowers users to bridge their Ethereum assets to Solana by connecting to Drift while providing the capability to trade on Drift directly from within the Metamask interface.

03 | Findings

Overall, we reported 3 findings.

We split the findings into **vulnerabilities** and **general findings**. Vulnerabilities have an immediate impact and should be remediated as soon as possible. General findings do not have an immediate impact but will help mitigate future vulnerabilities.

Severity	Count
Critical	0
High	0
Medium	0
Low	0
Informational	3

04 | General Findings

Here, we present a discussion of general findings during our audit. While these findings do not present an immediate security impact, they represent anti-patterns and may result in security issues in the future.

ID	Description
OS-DFT-SUG-00	The snap's dialog does not show certain important information regarding the transaction.
OS-DFT-SUG-01	Unpinned dependencies in the project introduce vulnerabilities to supply chain attacks focused on those specific dependencies.
OS-DFT-SUG-02	The RPC parameter types are not validated, which may result in type confusion issues.

OS-DFT-SUG-00 | Display Of Important Transaction Information

Description

The system utilizes Metamask's snap dialog feature to present users with specific transaction-related details before approving the transaction. However, certain crucial transaction information is currently not being displayed, such as:

- Changes in SOL/SPL token balances.
- The gas fee.

Remediation

Simulate the transaction to show this information so that the user is informed of these parameters before they decide to approve the transaction.

Patch

Drift Labs acknowledged the issue and will release a patch in subsequent versions.

OS-DFT-SUG-01 | Unpinned NPM Dependencies

Description

The application incorporates specific NPM packages for which the dependency versions are not explicitly defined in the project's configuration. This situation may allow supply chain attackers to exploit a package within the indicated version range. Consequently, developers who integrate these unpinned dependencies may unknowingly install the compromised package version.

Furthermore, it is crucial to consider the potential cascade effect of supply chain attacks. If a widely utilized package with numerous dependencies is compromised, many projects that employ that package may be impacted.

Remediation

Ensure that all the dependencies utilized within the application clearly specify the exact version required by the application.

Patch

Fixed in commit 50a9081 by making the following modification to the `package.json` file:

package.json DIFF

```diff
- "@typescript-eslint/eslint-plugin": "^6.2.0",
- "@typescript-eslint/parser": "^6.2.0",
- "eslint": "^8.44.0",
- "eslint-config-prettier": "^8.9.0",
- "husky": "^8.0.3",
- "lint-staged": "^13.2.3",
- "prettier": "^2.8.8",
- "typescript": "^5.1.6"
+ "@typescript-eslint/eslint-plugin": "6.2.0",
+ "@typescript-eslint/parser": "6.2.0",
+ "eslint": "8.44.0",
+ "eslint-config-prettier": "8.9.0",
+ "husky": "8.0.3",
+ "lint-staged": "13.2.3",
+ "prettier": "2.8.8",
+ "typescript": "5.1.6"
```

OS-DFT-SUG-02 | Type Confusion Issues

Description

The validation of RPC parameter types is currently lacking, which may expose the application to type confusion vulnerabilities due to the untrusted nature of the data passed through them.

Remediation

Add strict recursive checks against the type of each RPC parameter at run time since they are untrusted data.

```ts
if (typeof params.serializeConfig != 'object' &&
    typeof params.serializeConfig.requireAllSignatures != 'boolean' &&
    typeof params.serializeConfig.verifySignatures != 'boolean'
){
throw new Error("Type error");
}
```

Patch

Fixed in commit 50a9081 by adding strict type checks against all RPC parameters.

A | Vulnerability Rating Scale

We rated our findings according to the following scale. Vulnerabilities have immediate security implications. Informational findings can be found in the General Findings section.

Critical

Vulnerabilities that immediately lead to loss of user funds with minimal preconditions

Examples:

- Misconfigured authority or access control validation
- Improperly designed economic incentives leading to loss of funds

High

Vulnerabilities that could lead to loss of user funds but are potentially difficult to exploit.

Examples:

- Loss of funds requiring specific victim interactions
- Exploitation involving high capital requirement with respect to payout

Medium

Vulnerabilities that could lead to denial of service scenarios or degraded usability.

Examples:

- Malicious input that causes computational limit exhaustion
- Forced exceptions in normal user flow

Low

Low probability vulnerabilities which could still be exploitable but require extenuating circumstances or undue risk.

Examples:

- Oracle manipulation with large capital requirements and multiple transactions

Informational

Best practices to mitigate future security risks. These are classified as general findings.

Examples:

- Explicit assertion of critical internal invariants
- Improved input validation

B | Procedure

As part of our standard auditing procedure, we split our analysis into two main sections: design and implementation.

When auditing the design of a program, we aim to ensure that the overall economic architecture is sound in the context of an on-chain program. In other words, there is no way to steal funds or deny service, ignoring any chain-specific quirks. This usually requires a deep understanding of the program's internal interactions, potential game theory implications, and general on-chain execution primitives.

One example of a design vulnerability would be an on-chain oracle that could be manipulated by flash loans or large deposits. Such a design would generally be unsound regardless of which chain the oracle is deployed on.

On the other hand, auditing the implementation of the program requires a deep understanding of the chain's execution model. While this varies from chain to chain, some common implementation vulnerabilities include reentrancy, account ownership issues, arithmetic overflows, and rounding bugs.

As a general rule of sum, implementation vulnerabilities tend to be more "checklist" style. In contrast, design vulnerabilities require a strong understanding of the underlying system and the various interactions: both with the user and cross-program.

As we approach any new target, we strive to get a comprehensive understanding of the program first. In our audits, we always approach targets with a team of auditors. This allows us to share thoughts and collaborate, picking up on details that the other missed.

While sometimes the line between design and implementation can be blurry, we hope this gives some insight into our auditing procedure and thought process.