KUDELSKI SECURITY

Secure Code Review

Findings and Recommendations Report Presented to:

OptiFi

July 13, 2022
Version: 1.4

Presented by:

Kudelski Security, Inc.
5090 North 40th Street, Suite 450
Phoenix, Arizona 85018

For Public Release

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

EXECUTIVE SUMMARY

Overview

OptiFi engaged Kudelski Security to perform a secure code assessment.

The assessment was conducted remotely by the Kudelski Security Team. Testing took place from May 11th, 2021- June 3rd, 2022, and focused on the following objectives:

- Provide OptiFi with an assessment of the security of recently added functionality.
- To provide a professional opinion on code maturity, efficiency, and coding practices.
- To identify potential issues and include improvement recommendations based on the result of our tests.

This report summarizes the engagement, tests performed, and findings. It also contains detailed descriptions of the discovered vulnerabilities, steps the Kudelski Security Team took to identify and validate each issue, as well as any applicable recommendations for remediation.

At the time of this report, all findings classified with a severity level of low, medium, or high were fixed and resolved. Any informational findings were resolved through code remediation or accepted as a known state.

Key Findings

The following are the major themes and issues identified during the testing period. These, along with other items, within the findings section, should be prioritized for remediation to reduce the risk they pose.

- Insecure Account Initializations
- Unsafe Math Operations

During the test, the following positive observations were noted regarding the scope of the engagement:

- The code is clean, concise, and uses many best security practices.
- Anchor framework usage is very consistent and follows the recommended syntax

Scope and Rules of Engagement

Kudelski performed a Secure Code Review for OptiFi. The source code was supplied through a private repository. All files rust files (ending in *.rs) in the repo were the targets in scope for the engagement. No additional systems or resources were in scope for this assessment.

TECHNICAL ANALYSIS & FINDINGS

During the Secure Code Review, we discovered 1 finding that had a high severity rating, as well as 1 of medium severity.

The following chart displays the findings by severity.

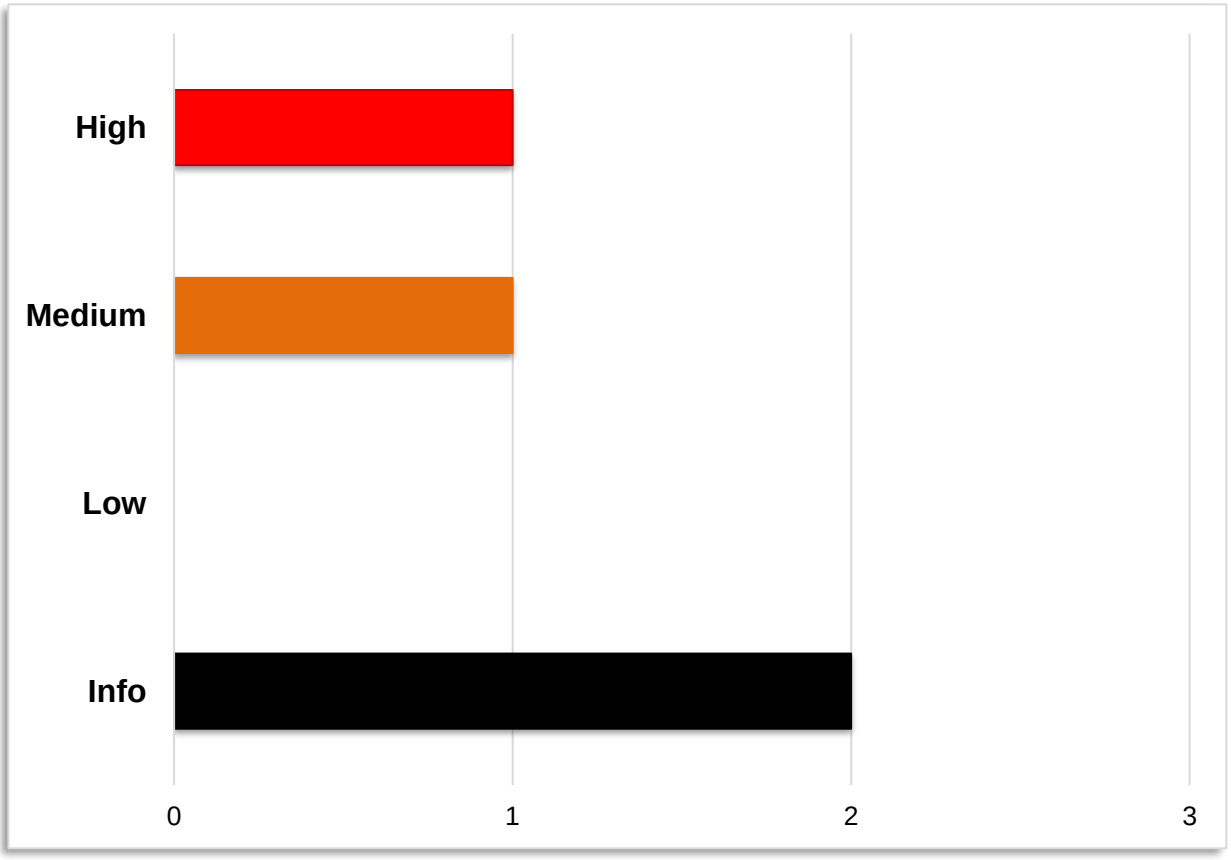

Figure 1: Findings by Severity

Findings

The *Findings* section provides detailed information on each of the findings, including methods of discovery, explanation of severity determination, recommendations, and applicable references.

The following table provides an overview of the findings.

#	Status	Severity	Description
1	Resolved	High	Insecure Account Initialization
2	Resolved	Medium	Dangerous Arithmetic – Underflow/Overflow
3	Resolved	Informational	Loss of precision
4	Accepted	Informational	Convert days to hours or minutes for Options settlement

Table 1: Findings Overview

KSI-01 – Insecure Account Initialization

Severity	HIGH

Impact	Likelihood	Difficulty
High	Medium	Medium

Description

The Kudelski security team discovered that margin_stress accounts can be reinitialized after they've already been initialized because the `init_if_needed` keyword in anchor prior to v0.24.x does not include an account discriminator for already initialized accounts.

KSI-02 – Dangerous Arithmetic – Underflow/Overflow

Severity	Medium

Impact	Likelihood	Difficulty
High	Medium	Medium

Description

The Kudelski security team discovered multiple instances of potential overflow and underflow conditions throughout the program's mathematical operations. Overflow and underflow conditions occur when integers are used in arithmetic operations where the result is a value outside of its range. In the case of the Rust programming language, these underflows and overflows create a panic at compile time, but in release mode, the Rust compiler silently ignores this behavior.

KSI-03 – Loss of precision

Severity	INFORMATIONAL

Impact	Likelihood	Difficulty
N/A	N/A	N/A

Description

The Kudelski Security team noticed that some critical parameters like prices and timestamps are large numbers that can round when converted to or from 32-bit floating-point.

For instance, below is the same Unix timestamp converted to f64, u64, f32, then from f32 to back to u64:

```
f64: 1654644628.3955457
u64: 1654644628
f32: 1654644600
u64: 1654644608
```

Example:
Rust Playground

KSI-04 – Convert days to hours for Options settlement and funds settlement and transfer

Severity	INFORMATIONAL

Impact	Likelihood	Difficulty
N/A	N/A	N/A

Description

The Kudelski Security team noticed and followed up with developers via conference call; that an epoch is used for Options settlements and funds transfer within the repository. The Solana SDK uses the concept of a dev epoch, while Anchor and Serum will utilize and subsequently have documentation around the UNIX epoch – which acts like a system clock, as well as the dev epoch.

METHODOLOGY

During this source code review, the Kudelski Security Services team reviewed code within the project within an appropriate IDE. During every review, the team spends considerable time working with the client to determine correct and expected functionality, business logic, and content to ensure that findings incorporate this business logic into each description and impact. Following this discovery phase, the team works through the following categories:

- Authentication
- Authorization and Access Control
- Auditing and Logging
- Injection and Tampering
- Configuration Issues
- Logic Flaws
- Cryptography

These categories incorporate common Solana vulnerabilities such as:

- Missing signer checks
- Missing ownership checks
- Missing rent exemption checks
- Signed invocation of unverified programs
- Solana account confusions
- Re-initiation with cross-instance confusion
- Arithmetic overflow/underflows
- Numerical precision errors
- Loss of precision in calculation
- Incorrect calculation
- Casting truncation
- Exponential complexity in calculation
- Insufficient SPL-Token account verification
- Unsafe design
- Any hidden backdoors
- Call stack depth
- Contract logic correctly implements the project specifications
- Flaws in design, logic, or access control
- Compiler warnings

These categories incorporate common Rust vulnerabilities such as:

- Unsafe Rust code: If a smart contract contains any unsafe Rust code, it may still suffer from memory corruptions such as buffer overflows, use after frees, uninitialized memory, etc.
- Outdated dependencies
- Redundant code

- Do not follow security best practices

Tools

The following tools were used during this portion of the test. A link for more information about the tool is provided as well.

- Visual Studio Code
- Semgrep
- Soteria
- Rudra
- Cargo-Audit
- Cargo-Geiger
- Cargo-Clippy

Vulnerability Scoring Systems

Kudelski Security utilizes a vulnerability scoring system based on the impact of the vulnerability, the likelihood of an attack against the vulnerability, and the difficulty of executing an attack against the vulnerability based on a high, medium, and low rating system

Impact
The overall effect of the vulnerability against the system or organization is based on the areas of concern or affected components discussed with the client during the scoping of the engagement.

> **High:**
> The vulnerability has a severe effect on the company and systems or has an effect within one of the primary areas of concern noted by the client
>
> **Medium:**
> It is reasonable to assume that the vulnerability would have a measurable effect on the company and systems that may cause minor financial or reputational damage.
>
> **Low:**
> There is little to no effect from the vulnerability being compromised. These vulnerabilities could lead to complex attacks or create footholds used in more severe attacks.

Likelihood
The likelihood of an attacker discovering a vulnerability, exploiting it, and obtaining a foothold varies based on a variety of factors, including compensating controls, location of the application, availability of commonly used exploits, and institutional knowledge

> **High:**
> It is extremely likely that this vulnerability will be discovered and abused
>
> **Medium:**
> It is likely that this vulnerability will be discovered and abused by a skilled attacker
>
> **Low:**
> It is unlikely that this vulnerability will be discovered or abused when discovered.

Difficulty
Difficulty is measured according to the ease of exploit by an attacker based on the availability of readily available exploits, knowledge of the system, and complexity of the attack. It should be noted that a LOW difficulty results in a HIGHER severity.

> **Low:**
> The vulnerability is easy to exploit or has readily available techniques for exploit
>
> **Medium:**
> The vulnerability is partially defended against, difficult to exploit, or requires a skilled attacker to exploit.
>
> **High:**
> The vulnerability is difficult to exploit and requires advanced knowledge from a skilled attacker to write an exploit

Severity
Severity is the overall score of the weakness or vulnerability as it is measured from Impact, Likelihood, and Difficulty

KUDELSKI SECURITY CONTACTS

NAME	POSITION	CONTACT INFORMATION
Brandon Gilchrist	**Blockchain Security Expert**	**brandon.gilchrist@kudelskisecurity.com**
Kelly Ryver	**Blockchain Security Expert**	**kelly.ryver@kudelskisecurity.com**