Security Assessment

Findings and Recommendations Report Presented to:

Aldrin – Review 2

September 24, 2021
Version: 1.0

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

Aldrin engaged Kudelski Security to perform a Security Assessment.

The assessment was conducted remotely by the Kudelski Security Team. Testing took place on September 15 - September 22, 2021, and focused on the following objectives:

- Provide the customer with an assessment of their overall security posture and any risks that were discovered within the environment during the engagement.
- To provide a professional opinion on the maturity, adequacy, and efficiency of the security measures that are in place.
- 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 Teams took to identify and validate each issue, as well as any applicable recommendations for remediation.

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 to the risk they pose.

- No apparent flaws where detected during the review.

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

- The team was very supportive and open to discuss the design choices made
- The mathematics were clear and precisly implemented.

Based on the account relationship graphs or reference graphs and the formal verification we can conclude that the reviewed code implements the documented functionality.

Scope and Rules Of Engagement

Kudelski performed an Security Assessment for Aldrin. The following table documents the targets in scope for the engagement. No additional systems or resources were in scope for this assessment.

The source code was supplied through private repositories at https://gitlab.com/crypto_project/defi/ammv2/-/tree/amm-v2-swap with the commit hash f21032bd2c9c946d4c8486ee5442a1bd39d05c52.

Files included in the code review	
ammv2/ ├── programs │ └── mm-farming-pool │ └── src │ ├── baskets.rs │ ├── curve │ │ ├── base.rs │ │ ├── calculator.rs │ │ ├── constant_product.rs │ │ ├── fees.rs │ │ └── mod.rs │ ├── spl_math │ │ ├── checked_ceil_div.rs │ │ ├── mod.rs │ │ ├── precise_number.rs │ │ └── uint.rs │ ├── farming.rs │ ├── fees.rs │ ├── lib.rs │ └── pool.rs	Rust implementation of the program

Table 1: Scope

TECHNICAL ANALYSIS & FINDINGS

During the Security Assessment, we discovered 1 finding that had an [Informational] severity rating.

The following chart displays the findings by severity.

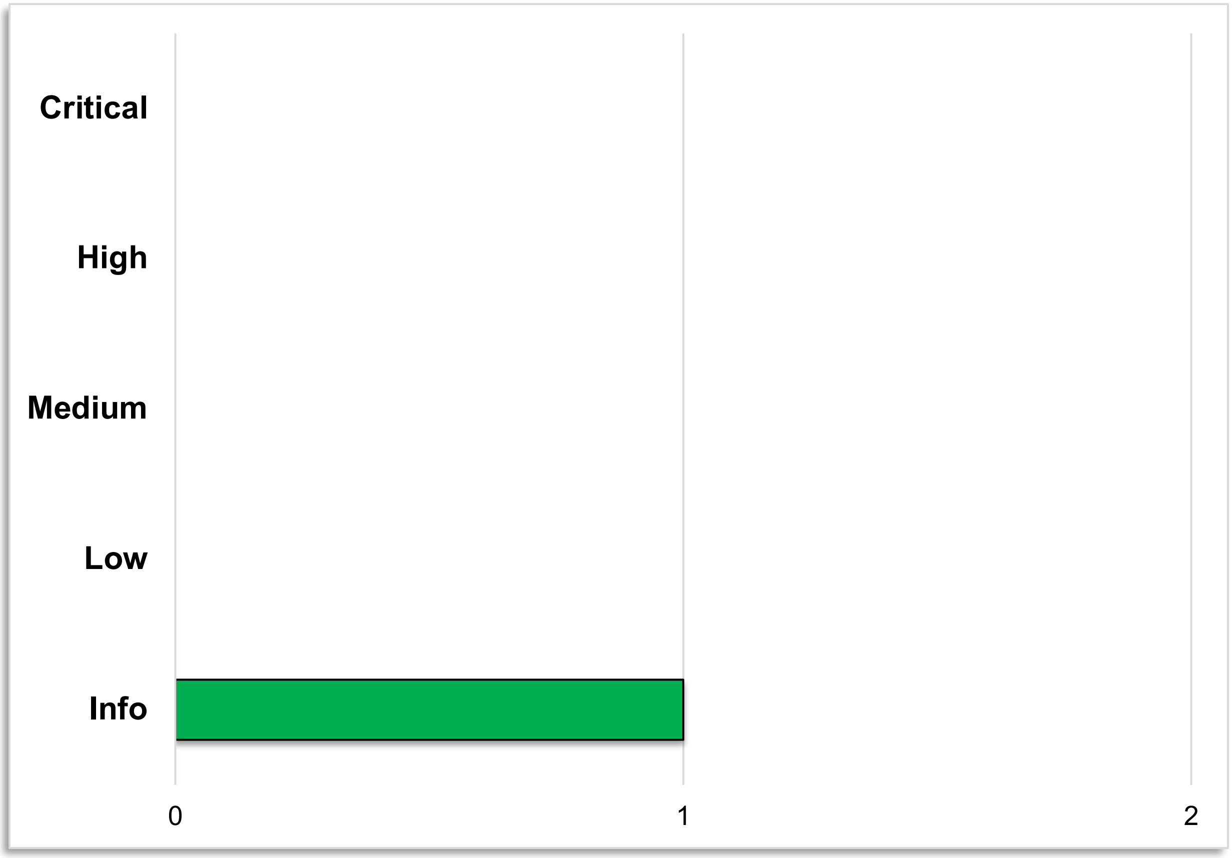

Figure 1: Findings by Severity

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.

#	Severity	Description
KS-CRYAI-2-F-01	**Informational**	Code contains copy of standard library

Table 2: Findings Overview

Technical analysis

Based on the source code the following account relationship graphs or reference graphs was made to verify the validity of the code as well as comfirmating that the intended functionality was implemented correctly and to the extent that the state of the repository allowed.

A number of further investigations were made which conluded that they did not pose a risk to the application. They were

- No potential panics were detected
- No potential errors regarding wraps/unwraps, expect and wildcards
- No internal unintentional unsafe references

Authorization

Normally a review use relationship graphs to show the relations between account input passed to the instructions of the program. The relations are used to verify if the authorization is sufficient for invoking each instruction. The graphs show if any unreferenced accounts exist. Accounts that are not referred to by trusted accounts can be replaced by any account of an attacker's choosing and thus pose a security risk.

In this case as the review concerned the mathematics of the application these graphs were omitted.

Conclusion

Based on the analysis of the mathematics and the formal verification we can conclude that the code implements the documented functionality to the extent of the code reviewed.

Technical Findings

General Observations

During the review the key functions were verified for mathematical accuracy.

Here we go through the code and verification of the mathematical implementations.

Verification of withdraw_single_token_type_exact_out functionality

This function implements the Balancer formula, used to calculate the number of pool tokens required to extract a single asset x in a balanced pool (two tokens, weighted at 0.5).

The function is contained in programs/mm-farming-pool/src/curve/constant_product.rs, from line 18:

```rust
pub fn withdraw_single_token_type_exact_out(
    source_amount: u128,
    swap_token_a_amount: u128,
    swap_token_b_amount: u128,
    pool_supply: u128,
    trade_direction: TradeDirection,
) -> Option<u128> {
    let swap_source_amount = match trade_direction {
        TradeDirection::AtoB => swap_token_a_amount,
        TradeDirection::BtoA => swap_token_b_amount,
    };
(*) let swap_source_amount = PreciseNumber::new(swap_source_amount)?;
    let source_amount = PreciseNumber::new(source_amount)?;
    let ratio = source_amount.checked_div(&swap_source_amount)?;
    let one = PreciseNumber::new(1)?;
    let base = one.checked_sub(&ratio)?;
    let root = one.checked_sub(&base.sqrt()?)?;
    let pool_supply = PreciseNumber::new(pool_supply)?;
(*) let pool_tokens = pool_supply.checked_mul(&root)?;
    pool_tokens.ceiling()?.to_imprecise()
}
```

The calculation is implemented step by step between the lines marked by (*). The calculation is performed using helper functions for precise arithmetic from spl_math, with the final result, the amount of pool tokens to be deposited, returned is an ordinary u128.

It is verified that the calculation corresponds to equation (23) in

https://balancer.fi/whitepaper.pdf#single-asset-withdrawal

Verification of product curve calculation

The swap calculation based on the constant product formula *xy=k*, where *x* and *y* are the quantities of two assets, is implemented in programs/mm-farming-pool/src/curve/constant_product.rs, from line 45:

```
    pub fn swap_without_fees(
        source_amount: u128,
        swap_source_amount: u128,
        swap_destination_amount: u128,
    ) -> Option<SwapWithoutFeesResult> {
        let invariant = swap_source_amount.checked_mul(swap_destination_amount)?;

        let new_swap_source_amount = swap_source_amount.checked_add(source_amount)?;
(*)        let (new_swap_destination_amount, new_swap_source_amount) =
            invariant.checked_ceil_div(new_swap_source_amount)?;

        let source_amount_swapped = new_swap_source_amount.checked_sub(swap_source_amount)?;
(*)      let destination_amount_swapped =
            map_zero_to_none(swap_destination_amount.checked_sub(new_swap_destination_amount)?)?;

        Some(SwapWithoutFeesResult {
            source_amount_swapped,
            destination_amount_swapped,
        })
    }
```

Here in the verification we will make the substitution swap_source_amount -> x, swap_destination_amount -> y, invariant -> k, swap_amonunt -> Δx

First the invariant k=xy is obtained. It should be that xy=(x+Δx)(y-Δy), where Δy is the amount of token y extracted, in the code destination_amount_swapped. As k is constant, Δy = y - k/(x+Δx). This is obtained sequentially in the code, in the lines marked by (*).

As the calculation is performed using checked arithmetic and appropriate error checks (e.g. map_zero_to_none) we can consider the function **verified**.

Code contains copy of standard library

Finding ID: KS-CRYAI-2-F-01

Severity: [**Informational**]

Status: [**Open]**

Description

The code repository contains a stripped down version of a standard library from the Solana Program Library (SPL).

Proof of Issue

```
ammv2/
├── programs
│   └── mm-farming-pool
│       └── src
│           ├── spl_math
│           │   ├── checked_ceil_div.rs
│           │   ├── mod.rs
│           │   ├── precise_number.rs
│           │   └── uint.rs
```

Copy of SPL Math library

Severity and Impact Summary

Copying code fdrom a standard library instead of referencing it poses a risk when it comes to maintainance. The risk is that any corrections to the code from the SPL will not be reflected in the code that has been copied. This puts the burden on the project to keep this code maintained.

Recommendation

Use the SPL code as a dependency and inherit the functionality and then create derivations of the code. This way the project can still rely on the SPL code being updated and at the same time build your own versions of the code.

References

N/A

METHODOLOGY

Kudelski Security uses the following high-level methodology when approaching engagements. They are broken up into the following phases.

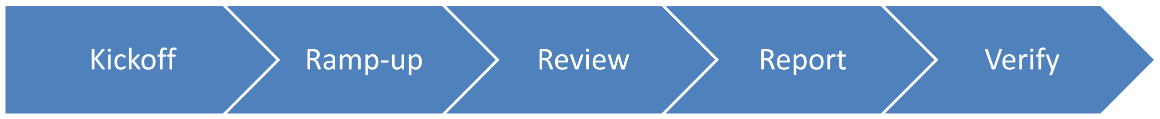

Figure 2: Methodology Flow

Kickoff

The project is kicked all of the sales process has concluded. We typically set up a kickoff meeting where project stakeholders are gathered to discuss the project as well as the responsibilities of participants. During this meeting we verify the scope of the engagement and discuss the project activities. It's an opportunity for both sides to ask questions and get to know each other. By the end of the kickoff there is an understanding of the following:

- Designated points of contact
- Communication methods and frequency
- Shared documentation
- Code and/or any other artifacts necessary for project success
- Follow-up meeting schedule, such as a technical walkthrough
- Understanding of timeline and duration

Ramp-up

Ramp-up consists of the activities necessary to gain proficiency on the particular project. This can include the steps needed for familiarity with the codebase or technological innovation utilized. This may include, but is not limited to:

- Reviewing previous work in the area including academic papers
- Reviewing programming language constructs for specific languages
- Researching common flaws and recent technological advancements

Review

The review phase is where a majority of the work on the engagement is completed. This is the phase where we analyze the project for flaws and issues that impact the security posture. Depending on the project this may include an analysis of the architecture, a review of the code, and a specification matching to match the architecture to the implemented code.

In this code audit, we performed the following tasks:

1. Security analysis and architecture review of the original protocol
2. Review of the code written for the project
3. Compliance of the code with the provided technical documentation

The review for this project was performed using manual methods and utilizing the experience of the reviewer. No dynamic testing was performed, only the use of custom built scripts and tools were used to assist the reviewer during the testing. We discuss our methodology in more detail in the following sections.

Code Safety

We analyzed the provided code, checking for issues related to the following categories:

- General code safety and susceptibility to known issues
- Poor coding practices and unsafe behavior
- Leakage of secrets or other sensitive data through memory mismanagement
- Susceptibility to misuse and system errors
- Error management and logging

This list is general list and not comprehensive, meant only to give an understanding of the issues we are looking for.

Technical Specification Matching

We analyzed the provided documentation and checked that the code matches the specification. We checked for things such as:

- Proper implementation of the documented protocol phases
- Proper error handling
- Adherence to the protocol logical description

Reporting

Kudelski Security delivers a preliminary report in PDF format that contains an executive summary, technical details, and observations about the project.

The executive summary contains an overview of the engagement including the number of findings as well as a statement about our general risk assessment of the project as a whole. We may conclude that the overall risk is low, but depending on what was assessed we may conclude that more scrutiny of the project is needed.

We not only report security issues identified but also informational findings for improvement categorized into several buckets:

- Critical

- High
- Medium
- Low
- Informational

The technical details are aimed more at developers, describing the issues, the severity ranking and recommendations for mitigation.

As we perform the audit, we may identify issues that aren't security related, but are general best practices and steps, that can be taken to lower the attack surface of the project. We will call those out as we encounter them and as time permits.

As an optional step, we can agree on the creation of a public report that can be shared and distributed with a larger audience.

Verify

After the preliminary findings have been delivered, this could be in the form of the approved communication channel or delivery of the draft report, we will verify any fixes withing a window of time specified in the project. After the fixes have been verified, we will change the status of the finding in the report from open to remediated.

The output of this phase will be a final report with any mitigated findings noted.

Additional Note

It is important to note that, although we did our best in our analysis, no code audit or assessment is a guarantee of the absence of flaws. Our effort was constrained by resource and time limits along with the scope of the agreement.

While assessment the severity of the findings, we considered the impact, ease of exploitability, and the probability of attack. These is a solid baseline for severity determination.

The Classification of identified problems and vulnerabilities

There are four severity levels of an identified security vulnerability.

Critical – vulnerability that will lead to loss of protected assets

- This is a vulnerability that would lead to immediate loss of protected assets
- The complexity to exploit is low
- The probablillty of exploit is high

High - A vulnerability that can lead to loss of protected assets

- All discrepancies found where there is a security claim made in the documentation that can not be found in the code
- All mismatches from the stated and actual functionality
- Unprotected key material

- Weak encryption of keys
- Badly generated key materials
- Tx signatures not verified
- Spending of funds through logic errors
- Calculation errors overflows and underflows

Medium - a vulnerability that hampers the uptime of the system or can lead to other problems

- Insecure calls to third party libraries
- Use of untested or nonstandard or non-peer-revied crypto functions
- Program crashes leaves core dumps or write sensitive data to log files

Low - Problems that have a security impact but does not directly impact the protected assets

- Overly complex functions
- Unchecked return values from 3rd party libraries that could alter the execution flow

Informational

- General recommendations

Tools

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

Tools used during the code review and assessment

- Rust – cargo tools
- IDE modules for Rust and analysis of source code
- Cargo audit which uses https://rustsec.org/advisories/ to find vulnerabilities cargo.

RustSec.org

About RustSec
The RustSec Advisory Database is a repository of security advisories filed against Rust crates published and maintained by the Rust Secure Code Working Group.

The RustSec Tool-set used in projects and CI/CD pipelines
‘cargo-audit’ - audit Cargo.lock files for crates with security vulnerabilities.
‘cargo-deny’ - audit `Cargo.lock` files for crates with security vulnerabilities, limit the usage of particular dependencies, their licenses, sources to download from, detect multiple versions of same packages in the dependency tree and more.

KUDELSKI SECURITY CONTACTS

NAME	POSITION	CONTACT INFORMATION