Security Assessment for AMMv2

Findings and Recommendations Report Presented to:

Aldrin

December 15, 2021

Version: 1.1.0

Presented by:

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

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

EXECUTIVE SUMMARY

Overview

Aldrin engaged Kudelski Security to perform a Security Assessment for AMMv2.

The assessment was conducted remotely by the Kudelski Security Team. Testing took place on October 18 - October 26, 2021, with a re-check following fixes, 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.

- KS-AMM2-01 – Stable swap verification

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

Based on formal verification we conclude that the reviewed code implements the documented functionality. As of the issuance of this report, all identified problems have been remediated to our satisfaction.

The issuance of the single template to change_fees is also reviewed and ready to be used on mainnet.

Scope and Rules of Engagement

Kudelski performed a Security Assessment for AMMv2. 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 a private repository at https://gitlab.com/crypto_project/defi/ammv2 with the commit hash c2629a43a6e800089423f3207034624de1915610.

In addition to the above, the following was reviewed to the “change fees” code

https://gitlab.com/crypto_project/defi/ammv2/-/commit/07aa985600e281c0ce37ba1ba5e9e0c1bf19d667

Files included in the code review
```
ammv2/
├── programs/
│   └── mm-farming-pool/
│       ├── src/
│       │   ├── cpi/
│       │   │   ├── dex.rs
│       │   │   └── mod.rs
│       │   ├── baskets.rs
│       │   ├── farming.rs
│       │   ├── fees.rs
│       │   ├── lib.rs
│       │   ├── orderbook.rs
│       │   └── pool.rs
│       ├── Cargo.toml
│       └── Xargo.toml
└── Cargo.toml
```

Table 1: Scope

TECHNICAL ANALYSIS & FINDINGS

During the Security Assessment for AMMv2, we discovered:

- 1 finding with HIGH severity rating.

The following chart displays the findings by severity.

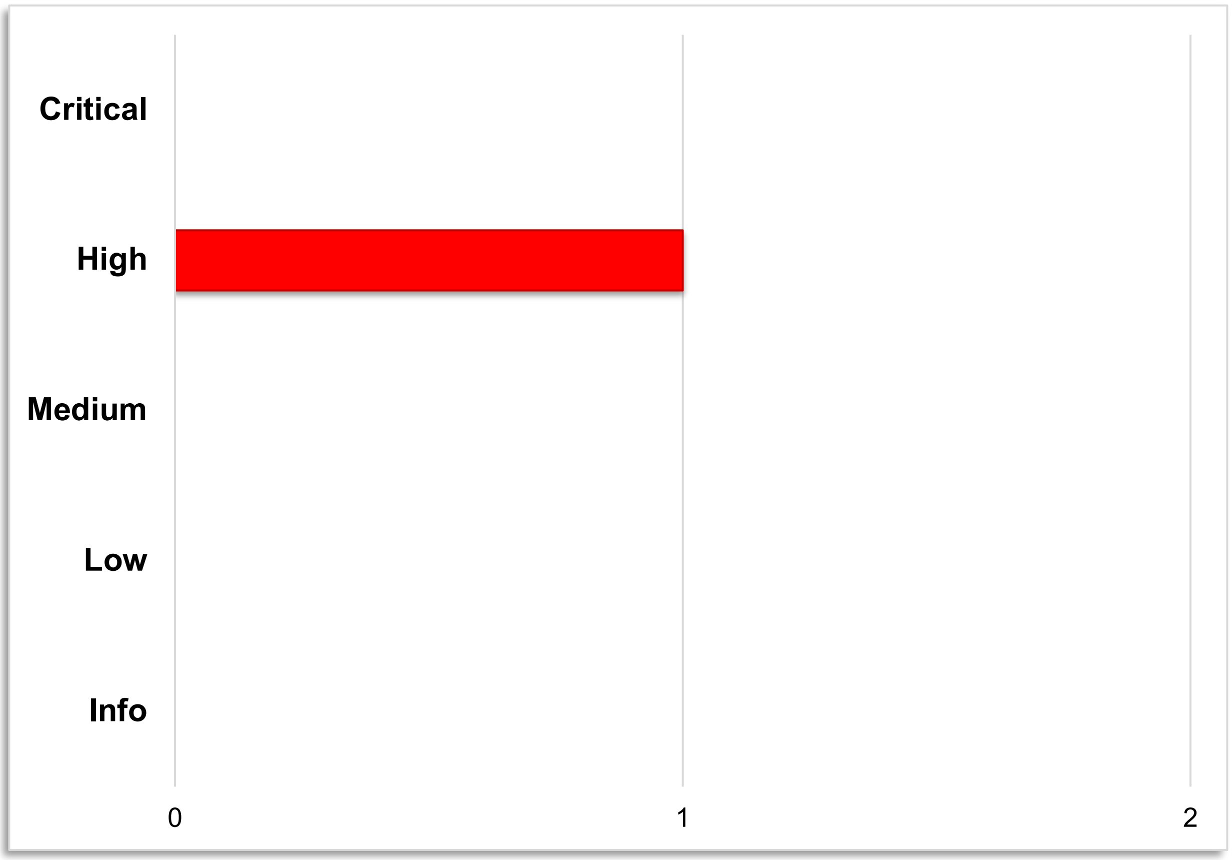

Figure 1: Findings by Severity

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-AMM2-01	**High**	Stable swap verification

Table 2: Findings Overview

Technical analysis

Based on the source code validity of the code as well as correct implementation of the intended functionality has been verified to the extent that the state of the repository allowed.

Further investigations were made which concluded 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

Based formal verification we conclude that the code implements the documented functionality to the extent of the code reviewed.

Technical Findings

General Observations

- `take_farming_snapshot` (l 260 `farming.rs`) -> time attribute of `farming_snapshot` changed, as described to the time stamp from the `farmingState` struct. The function itself checks-out and the time attribute change does not imply any critical changes on account structure and relations.
- `FarmingState` structs are built by the `initialize_farming` function, which takes an `InitializeFarming` struct as a parameter. The required checks are made on the `InitializeFarming` members, which are used to build the `FarmingState` instance.
- Changes to `withdraw_farmed` -> Changes do not affect account structures and relationships.

Stable swap verification

Finding ID: KS-AMM2-01
Severity: **High**
Status: **[Resolved]**

Description

The code aims to implements a AMM based on the stable swap invariant model. In this model, the balance of tokens in the pool must obey:

$$An^n \sum x_i + D = ADn^n + D_p$$

Here A is the amplification constant, n the number of different tokens in the pool, is the sum of the number of each token x_i in the pool when the prices are balance, Σx_i. This equation is obtained by introducing a dynamic leverage χ, given by:

$$\chi = \frac{A\Pi x_i}{(D/n)^n}$$

and defining D_p as follows:

$$D_p = \frac{D^{n+1}}{n^n \Pi x_i}$$

As the first equation is non-linear, to find D it is solved with a numerical method, such as the Newton-Raphsod interpolation:

$$D_{n+1} = D_n + \frac{f(D)}{f(D')} = \frac{D(nD_p + An^n \sum x_i)}{(n+1)D_p + D(An^n - 1)}$$

The code does not correctly implement the mathematical formalism above.

Proof of the issue

In the code this is implemented in `programs/mm-farming-pool/src/curve/stable.rs`.

Given a number of tokens of type a and tokens of type b, D is calculated by `fn compute_d`,

```
fn compute_d(leverage: u64, amount_a: u128, amount_b: u128) -> Option<u128> {
    let amount_a_times_coins = checked_u8_mul(&U256::from(amount_a),
N_COINS)?;
    let amount_b_times_coins = checked_u8_mul(&U256::from(amount_b),
N_COINS)?;
    let sum_x = amount_a.checked_add(amount_b)?; // sum(x_i), a.k.a S
    if sum_x == 0 {
        Some(0)
    } else {
        let mut d_previous: U256;
        let mut d: U256 = sum_x.into();

        // Newton's method to approximate D
        for _ in 0..32 {
            let mut d_product = d;
            d_product = d_product
                .checked_mul(d)?
                .checked_div(amount_a_times_coins)?;
            d_product = d_product
                .checked_mul(d)?
                .checked_div(amount_b_times_coins)?;
            d_previous = d;
            //d = (leverage * sum_x + d_p * n_coins) * d / ((leverage - 1) *
d + (n_coins + 1) * d_p);
            d = calculate_step(&d, leverage, sum_x, &d_product)?;
            // Equality with the precision of 1
            if almost_equal(&d, &d_previous)? {
                break;
            }
        }
        u128::try_from(d).ok()
    }
}
```

The loop in this function executes the Newton-Raphsod interpolation, where the update to D at each step is obtained by calling `fn calculate_step`,

```
fn calculate_step(initial_d: &U256, leverage: u64, sum_x: u128, d_product:
&U256) -> Option<U256> {
    let leverage_mul = U256::from(leverage).checked_mul(sum_x.into())?;
    let d_p_mul = checked_u8_mul(&d_product, N_COINS)?;

    let l_val = leverage_mul.checked_add(d_p_mul)?.checked_mul(*initial_d)?;

    let leverage_sub =
initial_d.checked_mul((leverage.checked_sub(1)?).into())?;
    let n_coins_sum = checked_u8_mul(&d_product, N_COINS.checked_add(1)?)?;

    let r_val = leverage_sub.checked_add(n_coins_sum)?;

    l_val.checked_div(r_val)
}
```

In these functions the equations described above are intended to be specific to the case `n=2`. However, in the notation of the code, `leverage=amp*N_COINS`, when in fact it should be `leverage=amp*N_COINS**N_COINS` = `leverage=amp*2**2` for `N_COINS=2`. Therefore, the calculation of `D` will be inaccurate.

A similar mistake is also found in `fn compute_new_destination_amount` (from line 102),

```

fn compute_new_destination_amount(
    leverage: u64,
    new_source_amount: u128,
    d_val: u128,
) -> Option<u128> {
    // Upscale to U256
    let leverage: U256 = leverage.into();
    let new_source_amount: U256 = new_source_amount.into();
    let d_val: U256 = d_val.into();

    // sum' = prod' = x
    // c =  D ** (n + 1) / (n ** (2 * n) * prod' * A)
    let c = checked_u8_power(&d_val, N_COINS.checked_add(1)?)?
        .checked_div(checked_u8_mul(&new_source_amount,
N_COINS_SQUARED)?.checked_mul(leverage)?)?;

    // b = sum' - (A*n**n - 1) * D / (A * n**n)
    let b = new_source_amount.checked_add(d_val.checked_div(leverage)?)?;

    // Solve for y by approximating: y**2 + b*y = c
    let mut y_prev: U256;
    let mut y = d_val;
    for _ in 0..32 {
        y_prev = y;
        y = (checked_u8_power(&y, 2)?.checked_add(c)?)
            .checked_div(checked_u8_mul(&y,
2)?.checked_add(b)?.checked_sub(d_val)?)?;
        if almost_equal(&y, &y_prev)? {
            break;
        }
    }
    u128::try_from(y).ok()
}
```

In particular, the calculations of b and c (lines 118 and 114 respectively) should end in `checked_div(N_COINS_SQUARED)` and `checked_mul(N_COINS_SQUARED)` but read `checked_div(N_COINS)` and `checked_mul(N_COINS)`.

Severity and impact summary

As the errors are in key equations for the calculations involving the pool, including the price of tokens and how many are exchanged in a transaction, the mathematics must be sound to avoid unfair trades or loss of funds due to calculation errors. This constitutes a high-to-critical severity vulnerability in the software, as it can lead to loss-of-funds or pricing problems (the equations with implementation errors are non-linear and solved using numerical methods, making it hard to estimate the severity of the impact on token economics).

Recommendation

We recommended that the mathematical error is solved as soon as possible. Mathematical derivations were provided by the audit team to assist with this and, as of the writing of the report, the issue has said to be solved by the developers.

References

- https://curve.fi/files/stableswap-paper.pdf

METHODOLOGY

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

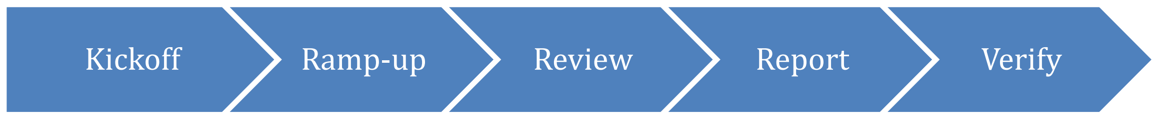

Figure 2: Methodology Flow

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 most 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